Manipulation of Elections: Algorithms and Infeasibility Results

by

Piotr Faliszewski

Submitted in Partial Fulfillment

of the

Requirements for the Degree

Doctor of Philosophy

Supervised by

Professor Lane A. Hemaspaandra

Department of Computer Science
The College
Arts and Sciences

University of Rochester
Rochester, New York

2008

To my parents, Bożena and Leszek Faliszewski.

Curriculum Vitae

Piotr Faliszewski was born on November 7th, 1980 in Kraków, Poland. Piotr's interests in computer science started in high school when he attended lectures at AGH University of Science and Technology in Kraków, was a Polish Children's Fund scholar, and participated in science camps organized by that Fund. During that time, in 1997, Piotr attended SciTech'97, a summer program for talented high school students organized by Israel Institute of Technology, Technion, in Haifa, Israel. Starting in 1999 Piotr attended AGH University of Science and Technology in Kraków, from which he graduated in 2004 with a Master of Science degree. In the Fall of 2004 Piotr joined the Ph.D. program of the Department of Computer Science at the University of Rochester, where he worked under the supervision of Professor Lane A. Hemaspaandra. In 2006 Piotr received a Master of Science degree from the University of Rochester. During each of his four summers in Rochester Piotr interned as an adjunct faculty member at the Rochester Institute of Technology.

Acknowledgments

I am deeply grateful to my advisor, Professor Lane A. Hemaspaandra, and to Professor Edith Hemaspaandra, who effectively became my coadvisor. Their advice, guidance, and encouragement have been most valuable and will forever influence how I see research and computer science.

I would like to thank Professors Saul Lubkin, Mitsunori Ogihara, Lenhart Schubert, and Joel Seiferas who, together with Lane and Edith, served on my thesis committee, freely offered their time and advice, and helped in improving this thesis. I also thank my undergraduate advisor, Dr. Janusz Jarosz, who was responsible for my original interest in computational complexity theory.

During my graduate student career I have worked closely with many wonderful researchers, both in the Rochester area and beyond. I am grateful to Yoram Bachrach, Eric Brelsford, Edith Elkind, Lane A. Hemaspaandra, Edith Hemaspaandra, Henning Schnoor, Ilka Schnoor, Mitsunori Ogihara, Jörg Rothe, Michael Wooldridge, and Michael Zuckerman for their work with me and for shaping my ideas on research and theoretical computer science. Many of the results presented in this thesis come from our joint projects.

Section 3.3, Chapter 4, and Section 5.1 describe my joint work with Edith Hemaspaandra and Lane A. Hemaspaandra (FHH06). In addition, parts of Chapter 4 are based on (Fal08). Section 5.2 describes my joint work with Eric Brelsford, Edith Hemaspaandra, Henning Schnoor, and Ilka Schnoor (BFH$^+$08). Sections 6.1 and 6.3 describe joint work with Edith Hemaspaandra, Lane A. Hemaspaandra, and Jörg Rothe (FHHR07; FHHR08). Section 6.2 describes joint work with Edith Hemaspaandra, and Henning Schnoor (FHS08). Chapters 1, 2, and 3 present the introduction and some preliminaries common to the whole thesis and, as such, are partially based on the just-mentioned papers.

I thank the Department of Computer Science at the University of Rochester. The department offered a wonderful environment for research and collaboration. I am grateful to the Computer Science Department at Heinrich-Heine-Universität Düsseldorf and to the Department of Computer Science at the University of Liverpool, both of which hosted visits by me. I am particularly grateful to the Computer Science Department at the Rochester Institute of Technology, which hired me as an adjunct faculty member for each of my summers in Rochester, provided a wonderful working environment, and greatly contributed to my development as an educator. I am particularly indebted to Professor Stanisław Radziszowski for many long conversations on research, politics, and everything.

Last but not least, I am very grateful to my family, and to my friends in Kraków, Poznań, and Rochester. Without their unconditional support this thesis would not have come to be.

Material in this thesis is based upon work supported by National Science Foundation under grant CCF-0426761. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the NSF or the University of Rochester.

Abstract

Voting and elections are at the core of democratic societies. People vote to elect leaders, decide policies, and organize their lives, but elections also have natural applications in computer science. For example, agents in multiagent systems often need to work together to complete some task, but each agent may have its own set of beliefs, preferences, and goals. Voting provides agents with a natural way to reach decisions that take all their preferences into account. With elections playing such an important role both in real-life political settings and in computer science, it is natural to ask about their resistance to misuse.

Two particular types of election misuse are manipulation and bribery. In manipulation, a group of voters chooses to misrepresent its preferences in order to obtain a more desirable outcome, and in bribery an outside agent, the briber, asks (possibly at a cost) a group of voters to change its votes, to obtain some outcome desirable for the briber. Classical results from political science show that, for any reasonable election system, there are scenarios where at least some voters have an incentive to attempt manipulation.

In this thesis we seek to protect elections from manipulators and bribers by making their computational task of finding good manipulations/bribes prohibitively expensive. When this is not possible, we seek to better understand (and even improve) the algorithmic attacks that manipulators and bribers can employ. In doing so, we develop new models of manipulation and bribery, and provide new approaches to studying the computational complexity of bribery and manipulation in elections.

Table of Contents

List of Figures

1 Introduction

Elections provide a natural way for a group of individuals, each of whom may have different goals and preferences, to arrive at a decision that is somehow acceptable to all of them. People vote on issues as varying as selecting their governments, choosing best movies, or deciding where to have dinner on a given day. Elections are also a very interesting and natural tool within computer science, especially in multiagent systems, where agents often have to make joint decisions. As just a few examples of applications of voting in multiagent systems we mention the work of Ephrati and Rosenschein (ER97) where elections are used for planning, the work of Ghosh et al. (GMHS99) where voting is used within a recommender system for movies, and the work of Dwork et al. (DKNS01) where elections are used to aggregate results from multiple web-search engines. With elections playing such an important role both in real-life political settings and in computer science, it is natural to ask about their resistance to misuse.

In this thesis we study the complexity of manipulation and bribery in elections. Manipulation refers to situations where voters choose to submit votes that do not truthfully reflect their real preferences and bribery refers to situations where it is an outside agent, the briber, that to achieve some goal asks a group of voters to change their votes (possibly at an expense). For an example of manipulation, a voter might rank his or her favorite candidate's strongest competitor as the least desired candidate, so that his or her favorite candidate's chances of winning might increase. In this thesis we are interested in computational questions regarding manipulation and bribery. For example, we ask if there is an efficient algorithm that, given an instance of an election, a preferred candidate in this election, and a set of manipulative voters, decides whether there are votes that the manipulative voters could cast to make their preferred candidate a winner. We seek to classify existing election systems with respect to the computational complexity of manipulation and bribery. In doing so, we develop new models of manipulation and bribery, and provide new approaches to studying computational complexity of bribery and manipulation in elections.

We stress that while manipulation and bribery have natural negative connotations, they do not necessarily need to correspond to cheating or any sort of illegal actions. For example, the problem of manipulation can be viewed as the problem of deciding whether a given candidate still has a chance of winning, given that some group of voters (the "manipulators") exercises its right to vote. If we know that no candidate other than the current winner can possibly become a winner, irrespective of what votes would be cast by those voters who haven't yet expressed their opinion, then we can safely terminate the election. Of course, voters in human elections would be scandalized if they were told that their votes were found to be irrelevant. However, in multiagent systems the fact that some agents do not need to vote means that both the election organizer and these agents save resources. This variant of the manipulation problem is sometimes referred to as the problem of vote elicitation.

In a similar spirit, one could view bribery as a problem of finding a coalition of voters who have the ability to switch the result of the election. One could view voters' prices as measuring the difficulty of convincing them to join the coalition.

1.1 Elections and Manipulating Them

The goal of an election system is to aggregate individual preferences and provide an outcome that "best represents" them. There is no clear definition of what *best represents* means and thus, throughout history, people have come up with quite a large number of different election systems. For example, most of the elections in the Athenian democracy were held using a simple majority rule: Members of the assembly could either agree with the speaker and vote *yes* or disagree with him and vote *no*. Such a system is both fair and natural but is also severely limited. Sometimes we need to choose among more than two alternatives!

One of the earliest accounts of this problem appears in a letter from Pliny the Younger to Titius Aristo, dated A.D. 105 (see (MU95) for a translation). The letter discusses a case where a group of liberated slaves, so-called freed men, was found near a dead body of the employer, consul Afranius Dexter. It was not apparent whether the consul committed suicide or whether he was killed by the freed men. Members of the Roman Senate were quick to divide themselves into three factions: Those who wanted the freed man to be left free (without, however, declaring them innocent), those who believed that the men should be banished and sent to an island, and those who believed that the men should be put to death. The Senate needed to vote in order to come up with a verdict. However, it was not at all clear whether the Senate should vote separately on each issue (in which case the men would be let free), should first decide between letting the men go free and punishing them somehow (in which case the joint ranks of the supporters of banishment and the supporters of the death penalty would have the majority), or should organize the voting in some other way. Eventually, the supporters of the death penalty dropped their proposal and joined the supporters of banishment. They worried that otherwise the freed men would go unpunished. Interestingly, Pliny the Younger, who was organizing the election, was himself an involved supporter of letting the men go free and insisted on voting on each issue separately.

The above case illustrates one of the most important questions regarding voting and election systems: How to organize elections in such a way that they are easy to conduct and neither the voters nor the organizers have the incentive (or ability) to skew the results. Death penalty supporters attempted to manipulate the election by voting for the banishment, even though it was against their true belief. The chair of the election, Pliny the Younger, attempted to *control* the result[1] via insisting that each issue should be voted on separately. Both the attempts succeeded partially: The men were punished but were not killed. The issue of manipulation (and its closely related cousin, bribery)

[1]Election control refers to settings where the organizers of the election change its structure (e.g., add or delete candidates, add or delete voters) in order to obtain a preferred outcome.

is at the heart of this thesis and the example described by Pliny the Younger shows how manipulation has been a serious threat to elections since the earliest days.

Could Pliny the Younger have avoided the problems of manipulation and control? For example, could he have chosen an election system in which the voters would have no incentive to misrepresent their votes? A most interesting sequence of results of Arrow (Arr63), Gibbard (Gib73), Satterthwaite (Sat75), and Duggan and Schwartz (DS00) answers this question negatively. Arrow showed that there is no election system that satisfies a small set of reasonable requirements and Gibbard and Satterthwaite showed that for any natural election system[2] there exists a scenario where some voter has an incentive to misrepresent his or her vote, i.e., to attempt manipulation. Strictly speaking, Gibbard and Satterthwaite showed their result for the so-called resolute election systems (i.e., systems that always elect a single unique winner) but Duggan and Schwartz (DS00) relaxed this resoluteness requirement.

Manipulation in elections is very tempting from the point of view of the voters involved, but from the point of view of the society it can be quite harmful. Let us consider the following example. We have three candidates, a, b, and c, and nine voters. Four voters, supporters of a, have sincere preferences expressed by $a > b > c$. That is, they like a best, then b, and then c. The next three voters, supporters of b, have preference $b > a > c$. Finally, two voters have preference $c > b > a$. The election is conducted using Borda election system (that is, for each vote, each candidate receives 2 points for being ranked first, 1 point for being ranked second, and 0 points for being ranked last). Candidates with the most points win. If the voters reported their true preferences then a would get 11 points, b would get 12 points, and c would get 4 points. So, b would be the unique winner. However, b would only have 1 point of advantage over a and so his or her supporters could attempt manipulation via voting $a > c > b$. a's supporters might reason that a couple of extra points would not change c's performance but taking points away from b would make a a winner. However, if the supporters of

[2]By *natural* we mean here that the system involves at least three candidates, is nondictatorial—there is no single voter who solely, irrespective of anyone else, decides who is the winner—and that every candidate can possibly become a winner.

b also attempted manipulation and voted $b > c > a$ (using a similar justification) then the outcome of the election would be that c gets 11 points, a and b get 8 points each, and c is the unique winner. Naturally, it would not serve the society well as 7 out of 9 voters in fact believe that c is the least desirable candidate.

In this thesis we do not discuss issues such as *why* a particular group of voters would attempt manipulation (or even *how* these voters got together, or game-theoretic aspects of multiple groups of manipulators acting at the same time). We focus on a scenario where there is a single group of manipulators who already got together and decided which candidate's victory they would like to ensure. The purpose of the above example is to show that manipulation attempts can lead to very undesirable outcomes and thus we should do our best to render manipulation infeasible.

1.2 Computational Social Choice

Results of Arrow, Gibbard and Satterthwaite, and Duggan and Schwartz are quite discouraging. However, a brilliant idea, originating in a sequence of papers by Bartholdi, Tovey, Trick, and Orlin (BTT89a; BO91; BTT92) is that even though voters might in principle have an incentive to cheat in an election, the problem of finding an appropriate manipulative action (e.g., figuring out how to misrepresent one's vote) might be so computationally difficult that in practice the voters would not have the resources needed to cheat effectively, and thus would be prevented from manipulating elections. This idea of computational study of elections proved to be remarkably fruitful and, together with the realization that societies of agents in multiagent systems face similar issues to those faced by societies of people, lead to the development of a new field of study, *computational social choice*, which intersects with political science, economics, and computer science. We will not describe computational social choice in detail here, but instead we point the reader to an excellent survey by Chevaleyre et al. (CELM07). In this thesis we focus on the line of research within computational social choice that directly extends the work of Bartholdi, Orlin, Tovey, and Trick.

Since the publication of the seminal works of Bartholdi, Orlin, Tovey, and Trick, a large body of research has been dedicated to the study of computational properties of election systems. Particular interest was paid to the issues of manipulation (CS03; EL05b; CS06; CSL07; HH07; PR07; BFH$^+$08; FHS08; MPRZ08), vote elicitation (CS02; Con07; Wal08; XC08; PRVW08), bribery (FHH06; CFRS07; EHRS07; Fal08), and control (HHR07a; HHR07b; ENR08; FHHR08; MPRZ08).[3] Separately, complexity-theoretic analysis of elections showed that some very attractive election systems may be impractical to use. Early work in this direction was done by Bartholdi, Tovey, and Trick (BTT89b), and was followed by papers by Hemaspaandra, Hemaspaandra, and Rothe (HHR97) on the complexity of Dodgson elections, by Hemaspaandra, Spakowski, and Vogel (HSV05) on Kemeny elections, and by Rothe, Spakowski, and Young (RSV03) on Young elections. These papers show that for the election systems they study the problem of deciding whether a particular candidate is a winner is complete for parallel access to NP (a believed-to-be-difficult complexity class). We point the readers interested in this line of work to the survey by Faliszewski et al. (FHHR). (That survey also discusses issues of control, manipulation, and bribery, but focuses on the results of the authors and does not attempt to present a complete view of the field.)

In this thesis we classify election systems with respect to whether they allow for computationally tractable manipulation and bribery. A standard technique for showing that a particular election-related problem (say, manipulation or bribery) is computationally intractable is to show that it is NP-hard. This approach is taken in almost all of the papers on computational social choice cited above, and it is the approach that we take in this thesis. One of the justifications for using NP-hardness as a barrier against manipulation and control of elections is that in multiagent settings any attempts to influence the election's outcome are made by computationally bounded software agents that have neither human intuition nor the computational ability to solve NP-hard problems.

Recently, such papers as (CS06; PR07; HH; MPS08) (and, to some extent, (ZPR08)) have studied the frequency (or sometimes, probability weight) of correctness of heuristics for voting problems. Although this is a fascinating and important direction, it does not

[3]The above list of papers is not complete and is only given to indicated how active the field is. The contents of this thesis were originally presented in some of the papers cited above.

at this point remove the need to study worst-case hardness. Indeed, we view worst-case study as a natural prerequisite to a frequency-of-hardness attack: After all, there is no point in seeking frequency-of-hardness results if the problem at hand is in P to begin with. And if one cannot even prove worst-case hardness for a problem, then proving “average-case” hardness is even more beyond reach. Also, current frequency-of-hardness results have debilitating limitations (for example, being locked into specific distributions; depending on unproven assumptions; and adopting “tractability” notions that declare undecidable problems tractable and that are not robust under even linear-time reductions). These models are arguably not ready for prime time and fail to imply average-case polynomial runtime claims. (EHRS07; HH) provide discussions of some of these issues.

1.3 Organization of the Thesis

This thesis is organized as follows. In Chapter 2 we give preliminary definitions, notation, and conventions from computer science. In particular, we provide necessary concepts from computational complexity theory, present the decision problems we use in our NP-hardness proofs, and discuss algorithmic tools that we use, such us network flows and approximation algorithms. In Chapter 3 we describe the election systems that we study, our conventions for representing votes, the problems of bribery and manipulation, and results relating the complexity of bribery and manipulation problems. Chapters 4, 5, and 6 constitute the main body of the thesis. In Chapter 4 we focus on the issues of bribery in plurality voting and in related systems, in Chapter 5 we study the complexity of manipulation and bribery for an important family of election rules (so-called *scoring protocols*), and in Chapter 6 we focus on Copeland voting (an election rule with some of the most promising computational properties).

2 Preliminaries

In this section we review basic notions and results regarding algorithms and computational complexity theory. A reader familiar with the material presented in the textbooks of Papadimitriou (Pap94), Bovet and Crescenzi (BC93), Balcázar, Díaz, and Gabarró (BDG95; BDG90), and Cormen et al. (CLRS01) may safely skim over this chapter, paying attention only to the conventions that we establish for this thesis.

Generally, we use standard mathematical notation. In particular, we use $\mathbb{N}$ to denote the set $\{0, 1, 2, \ldots\}$ and $\mathbb{Z}$ to denote the set $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$, and if A is a finite set then we use $\|A\|$ to denote the cardinality of A.

2.1 Computational Complexity

Computational complexity theory seeks to classify problems with respect to resources (e.g., time or space) necessary to solve them in a particular computation model. In this thesis we use Turing machines as our model of a computer and focus on time complexity. With a few exceptions, in this thesis we focus on decision problems, i.e., problems that seek a yes/no answer. We define decision problems using the following, standard format (see (GJ79)):

Name:	Primes.
Given:	An odd, positive integer n.
Question:	Is n prime?

Formally, decision problems are encoded as languages. An alphabet Σ is a finite set of symbols and a string over Σ is a finite sequence of symbols from Σ. Σ^* denotes the set of all strings over Σ. A language L over an alphabet Σ is a subset of Σ^*. For a string x in Σ^*, by $|x|$ we mean the length of x.

A decision problem is modeled as a language over some alphabet Σ via uniquely encoding each *yes* instance of the problem as some string over Σ. Throughout this thesis, without loss of generality, we assume that each decision problem is encoded as a language over $\Sigma = \{0, 1\}$ in a natural, efficient way. In particular, we assume that all numbers, unless specified otherwise, are encoded in binary. However, aside from this one convention, we do not look at specific details of our encodings. We point the reader to the classic textbook of Garey and Johnson (GJ79) for a discussion of this issue.

We often use terms "decision problem" and "language" interchangeably, even though, formally, these are two different types of entities. The reader can easily translate between decision problems and corresponding languages.

The main goal of this thesis is to classify various election problems either as easy or as hard; our primary tools to achieve such classification are the class P and the theory of NP-completeness. A language L belongs to P if it can be solved in polynomial time on a deterministic Turing machine (i.e., if there exists a polynomial-time algorithm that for each string $x \in \Sigma^*$ decides whether $x \in L$). A language L belongs to NP if it can be solved in polynomial time on a *nondeterministic* Turing machine.[1] Naturally, P $\subseteq$ NP.

To show that a language is computationally hard, we prove that some known-to-be-hard language *reduces* to it. At the intuitive level, a language A reduces to a language B if the ability to solve instances of B implies the ability to solve instances of A. Formally,

[1]Intuitively, NP is the class of problems whose *yes* instances have short (polynomially bounded in the instance length) proofs that they are in fact *yes* instances. Consider problem Clique below.

Name:	Clique.
Given:	Graph G and a nonnegative integer k.
Question:	Does G contain a clique of size k?

Clique is in NP because, given a graph G and a size-k subset of its vertices, it is easy to verify in polynomial time whether these vertices form a clique.

to establish the relative complexity of two languages we use the standard notion of a polynomial-time many-one reduction.

Definition 2.1. *Let A and B be two languages over alphabet Σ. $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ (A many-one reduces to B) if there is a polynomial-time computable function f such that*

$$(\forall x \in \Sigma^*)[x \in A \iff f(x) \in B].$$

A language is NP-hard if every language in NP reduces to it. A language is NP-complete if it is NP-hard and belongs to NP. Typically, to show that a problem is NP-complete we first prove that it is in NP, and then we give a polynomial-time many-one reduction to it from one of the well-known NP-complete problems. In Section 2.2 we will present two NP-complete problems that we use most often (almost exclusively) in our NP-completeness proofs.

Aside from the standard polynomial-time many-one reductions, in one of our results relating the complexity of manipulation and bribery we also need disjunctive truth-table reductions.

Definition 2.2. *Let A and B be two languages over alphabet Σ. We say that $A \leq_{\mathrm{dtt}}^{\mathrm{p}} B$ (A disjunctively truth-table reduces to B) if there is a polynomial-time procedure that on input x outputs a list of strings $y_1, \ldots, y_m$ such that $x \in A$ if and only if at least one of y_i, $1 \leq i \leq m$, is in B.*

When clear from context, we will use terms “reduce” and “reduction” as shorthands for “polynomial-time many-one reduce” and “polynomial-time many-one reduction.” Detailed treatment of various reduction types can be found, e.g., in the work of Ladner, Lynch, and Selman (LLS75).

As a convention, when giving an algorithm for a decision problem or when giving a reduction we assume that the inputs satisfy the constraints specified in the *Given* clause of the problem definition. For example, if we were to construct a reduction for the problem Primes, we would assume that the inputs to be handled are odd integers exclusively. Our reduction would not have to explicitly test if, for example, the input integer is even. Such semantic constraints on problem inputs can easily lead to strange situations (e.g., imagine the problem of satisfiability of boolean formulas where we assume that our input formulas are satisfiable), but in our problem definitions we will only use easily verifiable conditions, such as requiring numbers to be odd or even. Thus, all of our algorithms and reductions can be adjusted to verify whether their inputs satisfy appropriate conditions and, when not, either to outright reject (the algorithms) or to output fixed *no* instances (the reductions; we will never reduce to Σ^* so this will always be possible). This convention allows us to make our NP-completeness proofs a little clearer.

2.2 Two Useful NP-Complete Problems

Almost all NP-hardness results of this thesis follow via reductions from (variants of) Partition (in the case of weighted elections) and X3C (in the unweighted cases). Naturally, both Partition and X3C are NP-complete (see, e.g., (GJ79)). Partition asks whether it is possible to split a sequence of integers into two subsequences that have equal sums.

Name:	Partition.
Given:	A sequence $s_1, \ldots, s_n$ of nonnegative integers satisfying $\sum_{i=1}^{n} s_i \equiv 0 \pmod 2$.
Question:	Is there a set $A \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in A} s_i = \sum_{i \in \{1,\ldots,n\} - A} s_i$?

To prove one of our results in Section 5, we need a more restrictive version of the problem. Let $s_1, \ldots, s_n$ be a sequence of nonnegative integers such that $\sum_{i=1}^{n} s_i \equiv 0 \pmod 2$. In Partition′ we assume that for each i, $1 \leq i \leq n$, it holds that

$$s_i \geq \frac{2}{2+n} \sum_{i=1}^{n} s_i \tag{2.1}$$

and we ask whether there exists an $A \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in A} s_i = \frac{1}{2} \sum_{i=1}^{n} s_i$. For the sake of completeness, we include a proof that Partition′ remains NP-complete.

Lemma 2.3. *Partition′ is* NP*-complete.*

Proof. Clearly, Partition′ is in NP. We will show, by a reduction from the standard variant of Partition, that it is also NP-hard.

Let $q = (s_1, \ldots, s_n)$ be a sequence of nonnegative integers and let $2S = \sum_{i=1}^{n} s_i$. First, we construct a sequence $q' = (s'_1, o'_1, \ldots, s'_n, o'_n)$ of $2n$ nonnegative integers that has the following two properties. (1) q' can be partitioned into two equal-sum subsequences if and only if q can be. (2) Each partition of q' into two equal-sum subsequences splits q' into two sequences of the same cardinality. We define s'_i and o'_i, for $1 \leq i \leq n$, as follows.

$$\begin{aligned} s'_i &= 3^{i-1} + 3^n s_i. \\ o'_i &= 3^{i-1}. \end{aligned}$$

Sequence $s'_1, o'_1, \ldots, s'_n, o'_n$ sums up to $2S'$, where

$$S' = \frac{1}{2}\sum_{i=1}^{n}(s'_i + o'_i) = 3^n S + \sum_{i=1}^{n} 3^{i-1} = 3^n S + \frac{3^n - 1}{2}.$$

Clearly, any partition of $s'_1, o'_1, \ldots, s'_n, o'_n$ into two equal-sum subsequences splits q' into two halves such that if s'_i belongs to one then o'_i belongs to the other. It is also immediate that q can be partitioned into two equal-sum subsequences if and only if q' can.

To satisfy condition (2.1) we add a constant to each s'_i and o'_i. Define $\widehat{q}$ to be a sequence of numbers $\widehat{s}_1, \widehat{o}_1, \ldots, \widehat{s}_n, \widehat{o}_n$ such that for each i, $1 \leq i \leq n$,

$$\begin{aligned} \widehat{s}_i &= s'_i + S' \text{ and} \\ \widehat{o}_i &= o'_i + S'. \end{aligned}$$

Clearly, any partition of q' still is a partition of $\widehat{q}$, since any partition of q' splits q' into two subsequences of the same cardinality. The converse holds because any partition of $\widehat{q}$ has to split it into subsequences that each sum up to $\widehat{S} = S' + nS'$ and this is only possible if each subsequence contains exactly n elements. (A sum of more than n elements would be greater than $(n+1)S'$ and that would be more than the other subsequence could sum up to.) It remains to show that (2.1) holds for $\widehat{q}$. This is the case because each $\widehat{s}_i$ and $\widehat{o}_i$ is greater than S' and $S' = \frac{2}{2+2n}\widehat{S}$. (Note that sequence $\widehat{q}$ has $2n$ elements.) Since $\widehat{q}$ can be computed in polynomial time, the proof is completed. ❑

X3C, exact cover by 3-sets (our problem of choice when reducing to unweighted election problems) is defined as follows.

Name:	X3C.
Given:	A set $B = \{b_1, \ldots, b_{3k}\}$, $k \geq 1$, and a family of sets $\mathcal{S} = \{S_1, \ldots, S_n\}$ such that for each i, $1 \leq i \leq n$, it holds that $S_i \subseteq B$ and $\|S_i\| = 3$.
Question:	Is there a set $A \subseteq \{1, \ldots, n\}$, $\|A\| = k$, such that $\bigcup_{i \in A} S_i = B$?

The set A about which we ask in X3C is called an *exact cover of* B. It is a "cover" because every member of B belongs to some S_i such that $i \in A$, and it is "exact" because each member of B belongs to exactly one S_i such that $i \in A$.

2.3 Graphs and Flow Networks

An *undirected graph* G is a pair $(V(G), E(G))$, where $V(G)$ is the set of vertices and $E(G)$ is the set of edges, and each edge is an unordered pair of distinct vertices.[2] A *directed graph* is defined analogously, except that the edges are represented as ordered pairs. For example, if u and v are distinct vertices in an undirected graph G then G either has an edge $e = \{u, v\}$ that connects u and v or it doesn't. On the other hand, if G is a directed graph then G either has an edge $e' = (u, v)$ from u to v, or an edge $e'' = (v, u)$ from v to u, or both e' and e'', or neither e' nor e''.

For a directed graph G, the *indegree* of a vertex $u \in V(G)$ is the number of G's edges that enter u (i.e., the number of edges of the form (v, u) in $E(G)$). Similarly, the *outdegree* of $u \in V(G)$ is the number of edges that leave u (i.e., the number of edges of the form (u, v) in $E(G)$).

Intuitively, a flow network is a network of nodes with directed edges through which we want to transport some amount of flow from the source to the sink (where these are two designated nodes). Each edge e can carry up to $c(e)$ units of flow, and transporting each unit of flow through e costs $a(e)$. In the min-cost-flow problem we have a target flow value F, and the goal is to find a way of transporting F units of flow from the

[2]In this thesis, the symbols E and V are generally reserved for elections and voters, except for the just introduced "overloading" of them as $E(G)$ and $V(G)$ to mean sets of edges and vertices in a graph G. The intended meaning of E and V will always be clear from the context.

source to the sink, while minimizing the cost. (If there is no way of achieving target flow F, the cost in effect is infinite.) Let us now define flow networks formally.

Definition 2.4. *1. A* flow network *is a quintuple* (K, s, t, c, a)*, where* K *is a set of nodes that includes the* source s *and the* sink t*,* $c : K^2 \to \mathbb{N}$ *is the* capacity function*, and* $a : K^2 \to \mathbb{N}$ *is the* cost function*. We assume that* $c(u, u) = a(u, u) = 0$ *for each node* $u \in K$*, and that at most one of* $c(u, v)$ *and* $c(v, u)$ *is nonzero for each pair of distinct nodes* $u, v \in K$*. We also assume that if* $c(u, v) = 0$ *then* $a(u, v) = 0$ *as well.*

2. Given a flow network (K, s, t, c, a)*, a* flow *is a function* $f : K^2 \to \mathbb{Z}$ *that satisfies the following conditions:*

(a) For each $u, v \in K$*, we have* $f(u, v) \leq c(u, v)$*, i.e., capacities limit the flow.*

(b) For each $u, v \in K$*, we have* $f(u, v) = -f(v, u)$*.*[3]

(c) For each $u \in K - \{s, t\}$*, we have* $\sum_{v \in K} f(u, v) = 0$*, i.e., the flow is conserved in all nodes except the source and the sink.*

3. The value of flow f *is:*

$$\mathit{flowvalue}(f) = \sum_{v \in K} f(s, v).$$

The particular flow network we have in mind will always be clear from the context and so we will not indicate it explicitly.

4. The cost of flow f *is defined as:*

$$\mathit{flowcost}(f) = \sum_{u,v \in K} a(u, v) f(u, v).$$

That is, we pay the price $a(u, v)$ *for each unit of flow that passes from node* u *to node* v*.*

[3]Note that each flow is fully defined via its nonnegative values. Whenever we speak of a flow (e.g., when defining some particular flows) we will just speak of its nonnegative part.

Given a flow network (K, s, t, c, a) we will often refer to pairs of distinct nodes $(u, v) \in K^2$ such that $c(u, v) > 0$ as edges.

Below we define the min-cost-flow problem, which is well-known from the literature. The definition we employ here is not the most general one and the reader might want to consult, for example, the monograph by Ahuja, Magnanti, and Orlin (AMO93) for a comprehensive discussion of the problem.

Definition 2.5. *We define the* min-cost-flow problem *as follows: Given a flow network $N = (K, s, t, c, a)$ and a target flow value F, find a flow f that has value F (if one exists) and has minimum cost among all such flows, or otherwise indicate that no such flow f exists.*

The min-cost-flow problem has a polynomial-time algorithm.[4] There is a large body of work devoted to flow problems and we will not even attempt to provide a complete list of references here. Instead, we point the reader to the excellent monograph by Ahuja, Magnanti, and Orlin (AMO93), which provides descriptions of polynomial-time algorithms, theoretical analysis, and numerous references to previous work on flow-related problems. We also mention that the issue of flows is so prevalent in the study of algorithms that the textbook of Cormen et al. contains an exposition of the min-cost-flow problem (CLRS01, p. 787).

2.4 Approximation Algorithms

If a given problem is NP-complete then, unless P = NP, there is no polynomial-time algorithm for this problem. However, this does not mean that the problem is necessarily hard to solve in practice. It might be the case that an interesting subproblem is easy, or that typically the interesting instances are easy to solve, or that there are

[4]The Min-cost-flow problem is often defined in terms of capacity and cost functions that are not necessarily limited to nonnegative integer values and so the corresponding flows are not restricted to integer values either. However, crucially for us, it is known that if the capacity and cost functions have integral values (as we have assumed) then there exist optimal solutions to the min-cost-flow problem that use only integer-valued flows and that can be found in polynomial time.

fast approximation algorithms for the problem. In this thesis we will be interested in this last possibility. We warn the reader that this subsection is geared strictly to the use of approximation algorithms in this thesis. Thus, we only introduce those notions that are necessary for us here and define these generally enough only as to be useful for our purposes. We point the readers interested in a more thorough treatment to the textbooks of Vazirani (Vaz03) and Ausiello et al. (ACG$^+$99).

Approximation algorithms are typically considered in the context of the so-called search problems (or optimization problems), that is, problems where the goal is to find a possible solution (the goal is to find a high-quality solution) instead of just deciding whether one exists. For example, in a decision variant of Partition we are given a sequence of nonnegative integers $s_1, \ldots, s_n$ such that $\sum_{i=1}^{n} s_i \equiv 0 \pmod 2$, and the question: Is there a subsequence that sums up to exactly $S = \frac{1}{2}\sum_{i=1}^{n} s_i$? In the search variant of Partition we would want to compute such a subsequence. An approximate solution would be a subsequence that sums up to some value S', $S' \leq S$, and we would measure the quality of such a solution via asking for which rational ε, $0 < \varepsilon < 1$, it holds that $S' \geq (1-\varepsilon)S$.

Let $\mathcal{S}$ be a search problem. For each instance I of $\mathcal{S}$ we let $Sol(I)$ mean the set of all feasible solutions. Without loss of generality we assume that $Sol(I)$ is never empty (for example, we can include a dummy solution for each instance). We let f be a function such that for each instance I and solution $s \in Sol(I)$, $f(I,s)$ is a nonnegative integer. We interpret $f(I,s)$ as the quality of solution s for instance I. In maximization problems we seek a solution s with as high a value of $f(I,s)$ as possible, and in minimization problems we seek a solution s with as low a value of $f(I,s)$ as possible. f is sometimes called the goal function.[5] By $Opt(I)$ we mean the value of the best solution for I, that is, for the case of maximization problems, $Opt(I) = \max\{f(I,s) \mid s \in Sol(I)\}$ and for the case of minimization problems, $Opt(I) = \min\{f(I,s) \mid s \in Sol(I)\}$.

We say that an algorithm $\mathcal{A}$ is a *fully polynomial-time approximation scheme* (FP-

[5]For many search problems the choice of the goal function is obvious. However, this is not always the case. In particular, we will run into the problem of selecting an appropriate goal function when discussion approximate solutions for manipulation, in Section 5.2.

TAS) for $\mathcal{S}$ if it holds that for each instance I of $\mathcal{S}$ and each rational ε, $0 < \varepsilon < 1$, $\mathcal{A}$ computes, in time polynomial in the size of I and $\frac{1}{\varepsilon}$, a solution $\mathcal{A}(I, \varepsilon)$ such that

1. for a maximization problem, $f(\mathcal{A}(I, \varepsilon)) \geq (1 - \varepsilon) Opt(I)$,
2. for a minimization problem, $f(\mathcal{A}(I, \varepsilon)) \leq (1 + \varepsilon) Opt(I)$.

An FPTAS allows one to flexibly balance the amount of time that one is willing to invest in computing a solution, and the quality of the solution that one expects. Unfortunately, unless P = NP, not all NP-complete problems have an FPTAS. For example, consider an NP-complete problem whose search variant has the property that for each instance I, $Opt(I)$ is polynomially bounded in the size of I. (It is easy to give natural examples of such problems.) If such a problem had an FPTAS, then, via computing an appropriately good approximation, one could compute an optimal solution to each of this problem's instances in polynomial time.

3 Elections, Bribery, and Manipulation

In this chapter we define our models of elections, manipulation, and bribery, and present some relations between various flavors of manipulation and bribery.

3.1 Elections and Election Systems

We view an election as a pair $E = (C, V)$, where $C = \{c_1, \ldots, c_m\}$ is a finite set of candidates[1] and V is a finite collection of voters $v_1, \ldots, v_n$. Each voter has some preference regarding the candidates, but the way the voters specify their preferences depends on the setting. We focus on the so-called *rational-voter* model, but occasionally we also consider other models, such as the *irrational-voter* model. The rational-voter and irrational-voter models are defined as follows.

Rational voter model. Each voter's preference is represented as a linear order over the candidate set.[2] That is, each voter v_i has a *preference list* $c_{i_1} > c_{i_2} > \cdots > c_{i_m}$,

[1]Note that, technically, we allow C to be an empty set but in this thesis we ignore such cases. However, we mention that elections with no candidates sometimes arise naturally in scenarios regarding control of elections, in particular in control via (runoff) partition of candidates. See, e.g., (BTT92; HHR07a; FHHR07; FHHR08; ENR08).

[2]In this thesis we take "linear order" to mean a strict total order. This is a common convention within voting theory, see, for example, the book of Austen-Smith and Banks (AB00). However, we mention that the terminology typically used within mathematics allows linear orders to be nonstrict, i.e., to include ties.

with $\{i_1, i_2, \ldots, i_m\} = \{1, 2, \ldots, m\}$, which lists the candidates from the most preferred to the most despised. (We sometimes refer to preference lists as *preference orders.*) The rational voter model is the standard model in computational social choice literature and we adopt it as the default for this thesis. In many of our proofs we use the following notational convention.

Notation 3.1. *Within every election we fix some arbitrary order over the candidates. Any occurrence of a subset D of candidates in a preference list means the candidates from D are listed with respect to that fixed order. Occurrences of $\overrightarrow{D}$ mean the same except that the candidates from D are listed in the reverse order.*

For example, if $C = \{a, b, c, d, e\}$, with the alphabetical order being used, and $D = \{a, c, e\}$ then $b > D > d$ means $b > a > c > e > d$, and $b > \overrightarrow{D} > d$ means $b > e > c > a > d$.

Irrational voter model. Each voter's preferences are represented as a *preference table* that for every unordered pair of distinct candidates c_i and c_j in C indicates whether the voter prefers c_i to c_j (i.e., $c_i > c_j$) or prefers c_j to c_i (i.e., $c_j > c_i$). In the context of computational social choice, the irrational voter model was introduced by Faliszewski et al. (FHHR07) in their study of the complexity of Llull and Copeland election systems.

Other preference models include, for example, CP-nets introduced by Boutilier et al. (BBD$^+$04), and utility-based voting (see Section 3.1.1 for more references). Within computational social choice, CP-nets were considered, for example, by Xia et al. (XLY07).

Depending on the setting, in addition to their preferences, voters might also have several other attributes. For example, in the case of weighted elections each voter v_i, $1 \leq i \leq n$, has an integer *weight*, $\omega(v_i)$, and his or her vote counts as if it was cast by $\omega(v_i)$ weight-1 voters. (However, the voter may not choose to split the vote and use some part of the weight to vote in one way and some part to vote in another.) Given a subcollection U of voters, by $\omega(U)$ we mean the sum of their weights. We will often refer to $\omega(U)$ as "the vote weight of U" or "the total weight of U."

For the case of bribery, we often assume that each voter has a price that the briber has to pay to affect the vote. In our most typical setting the price of a voter v_i will simply be a nonnegative integer $\pi(v_i)$. As in the case of weights, for a subcollection of voters U, by $\pi(U)$ we mean the sum of their prices. In Section 4.2 (and, to some extent, in Section 6.3) we will study a more refined price model, but we defer its discussion until that section.

Unless we explicitly specify otherwise, by default we consider settings in which voters do not have weights and do not have prices.

In addition to having an election with a set of candidates and a collection of voters, we also need an election system, $\mathcal{E}$, that specifies which candidates are winners. Below we describe the election systems and families of election systems that we consider in this thesis. Following our convention, we assume the rational-voter model, but many of the systems below have natural interpretations in other models. For each of the systems that assigns points to candidates, those candidates with the most points are winners.

Scoring protocols. Let $E = (C, V)$ be an election with m candidates $c_1, \ldots, c_m$. A scoring protocol for m candidates is a vector $(\alpha_1, \ldots, \alpha_m)$ of nonnegative integers, with $\alpha_1 \geq \cdots \geq \alpha_m$. Each candidate $c_k \in C$ receives α_i points for each vote that ranks him or her at the ith place. Each particular scoring protocol α regards only a fixed number of candidates, but many election systems can be expressed as families of scoring protocols, $(\alpha^0, \alpha^1, \ldots, \alpha^m, \ldots)$, where each α^i is a scoring protocol $(\alpha^i_1, \alpha^i_2, \ldots, \alpha^i_i)$ for exactly i candidates.

Plurality. In *plurality* each voter gives one point to his or her favorite candidate. Plurality can be viewed as a family of scoring protocols of the form $(1, 0, \ldots, 0)$.

Borda. In *Borda* (sometimes also called *Borda count*), each voter v_i assigns each candidate c_j as many points as the number of candidates that v_i prefers c_j to. Borda can be viewed as a family of scoring protocols of the form $(m-1, m-2, \ldots, 0)$, where m is the number of candidates participating in a given election.

Veto. In *veto* each voter gives one point to each candidate but one (the vetoed one). Veto can be viewed as a family of scoring protocols of the form $(1, \ldots, 1, 0)$.

k-approval. In *k-approval* each voter gives one point to each of his or her top k candidates. k-approval can be viewed as a family of scoring protocols of the form $(\underbrace{1, \ldots, 1}_{k}, 0, \ldots, 0)$. Typically, we take "$k$" in k-approval to be a fixed constant but, via a slight abuse of notation, one could say that, for example, veto is $(m-1)$-approval. Generalizations of k-approval include elections where each voter assigns some points to the first k candidates on his or her preference list and all the other candidates receive no points from that particular voter. (The way in which the voters distribute points among the first k voters could be specified, e.g., via some scoring protocol.)

Approval. In *approval* each voter gives one point to each of the candidates that he or she approves of and gives zero points to all the other candidates. In the case of approval voting, we typically view voters' preferences not as preference lists but as 0/1 approval vectors, specifying which candidates receive points. (However, see the work of Brams and Sanver (BS06) for a variant of approval that incorporates both preference lists and approval vectors, and see the work of Erdélyi, Nowak, and Rothe (ENR08) for some computational results regarding this system.)

Copeland. *Copeland$^\alpha$ voting* (where α is a rational value such that $0 \leq \alpha \leq 1$) is somewhat different from the previous rules. Instead of having each voter distribute some number of points among the candidates, in Copeland$^\alpha$ points are distributed based on the results of the so-called head-to-head contests between the candidates. For each pair of candidates c_i, c_j we look at whether more voters prefer c_i to c_j or whether more voters prefer c_j to c_i. If more voters prefer c_i (i.e., c_i is the winner of this head-to-head contest) then c_i receives a point, if more voters prefer c_j (i.e., c_j is the winner of this head-to-head contest) then c_j receives the point, otherwise there is a tie, and both candidates receive α points each.[3]

[3] In the literature the term "Copeland elections" is most often used for the system Copeland$^{0.5}$ (e.g., (SM96; MS97) and a rescaled version of (CSL07)), but has occasionally been used for Copeland^0

Llull. Llull is a synonym for Copeland^1 voting.

Condorcet. A *Condorcet winner* is a candidate c_i that wins head-to-head contests against every other candidate. Naturally, in any election there is at most one Condorcet winner and it is possible for there to be none.

For each of the above systems (except Condorcet) by $score_E(c_i)$ we mean the number of points of candidate c_i in election E. In the case of Copeland$^\alpha$ we often write $score^\alpha_E(c_i)$ instead, to emphasize the tie-reward value α. In the case of elections based on head-to-head contests we are also interested in the results of these contests. Informally put, if $E = (C, V)$ is an election and if c_i and c_j are two candidates in C then by $\mathrm{vs}_E(c_i, c_j)$ we mean the surplus of votes that candidate c_i has over c_j. Formally:

$$\mathrm{vs}_E(c_i, c_j) = \begin{cases} 0 & \text{if } c_i = c_j \\ \|\{v \in V \mid v \text{ prefers } c_i \text{ to } c_j\}\| & \\ -\|\{v \in V \mid v \text{ prefers } c_j \text{ to } c_i\}\| & \text{otherwise.} \end{cases}$$

So: If c_i defeats c_j in a head-to-head contest in E then $\mathrm{vs}_E(c_i, c_j) > 0$; if they are tied then $\mathrm{vs}_E(c_i, c_j) = 0$; and if c_j defeats c_i then $\mathrm{vs}_E(c_i, c_j) < 0$. We can express the score of a candidate $c_i \in C$ in a Copeland$^\alpha$ election $E = (C, V)$ as

$$\begin{aligned} score^\alpha_E(c_i) &= \|\{c_j \in C \mid c_i \neq c_j \text{ and } \mathrm{vs}_E(c_i, c_j) > 0\}\| \\ &\quad + \alpha\|\{c_j \in C \mid c_i \neq c_j \text{ and } \mathrm{vs}_E(c_i, c_j) = 0\}\|. \end{aligned}$$

Note that the highest possible Copeland$^\alpha$ score in any election $E = (C, V)$ is $\|C\| - 1$.

Except for Condorcet rule, all the election systems presented above have the property that they always select at least one winner. However, they may also select more than one. Throughout this thesis, whenever we will ask whether a particular candidate is a winner (or, whether this candidate can become a winner via some action), we mean asking whether he or she is *one* of the winners (or, whether he or she can become *one* of the winners). This convention is typically referred to as the nonunique-winner model. In the *unique*-winner model, as expected, we would consider a candidate a winner only

(e.g., (PRK07; FHHR07)).

if he or she were *the only* winner. The literature on computational social choice is divided as to which winner model is more natural. Some authors use one, some use the other, and some use both. Here we adopt the nonunique-winner model because it is typically easier to work with, and in almost all cases (certainly in all cases studied in this thesis), hardness/easiness results regarding nonunique winners also hold for unique winners (and the other way round).

3.1.1 Utility-Based Voting

Sometimes, instead of using the rational-voter model, it is convenient to consider the so-called utility-based model. In the context of computational social choice it was introduced by Elkind and Lipmaa (EL05a), and was later picked up by Faliszewski (Fal08) and Meir et al. (MPRZ08). The idea of utility-based voting is that each voter has some number of points that are distributed freely among candidates (although we sometimes put a bound on the number of points that a voter can assign to a single candidate).

Definition 3.2. *Let e and b be two positive integers. By an (e, b)-*election *we mean an election over some candidate set $C = \{c_1, \ldots, c_m\}$, where each voter from the voter collection V distributes e integral points among the candidates, never assigning more than b points to a single candidate. The candidates with most points are the winners.*

*A free-form (e, b)-*election *is an (e, b)-*election *where voters can choose not to use all of their points.*[4]

Many election systems that we have described in the context of the rational-voter model can naturally be expressed as (e, b)-elections. For example, if m is the number of candidates then plurality can be expressed as $(1, 1)$-elections, k-approval as $(k, 1)$-elections, approval as free-form $(m, 1)$-elections, and veto as $(m - 1, 1)$-elections. (e, b)-elections with fairly large values of e and b (say, with e being polynomially related to the number of candidates and with $b = e$) are also very interesting as they capture the spirit of allowing the voters to express the intensity of their preference.

[4]Note that an (e, b)-election might be impossible, e.g., if e is too large or if we have too few candidates. On the other hand, free-form (e, b)-election is always possible.

3.2 Manipulating Elections

The term "manipulating elections" refers to any attempt (manipulation, bribery, control, etc.) to skew the result of an election. However, in this thesis we focus on only two such types of attempts, namely manipulation and bribery. Informally, in the bribery problem we are given an election $E = (C, V)$, a budget k, some distinguished candidate p, and we ask if we can ensure that p is a winner by changing preference lists of at most k voters. More formally, for an election system $\mathcal{E}$ we define the $\mathcal{E}$-bribery problem to be the following. (All our numbers are nonnegative integers.)

Name:	$\mathcal{E}$-bribery.
Given:	An election $E = (C, V)$, where each voter has a preference list over the candidates in C, a distinguished candidate $p \in C$, and a nonnegative integer k.
Question:	Is it possible to ensure that p is a winner of the $\mathcal{E}$ election E by changing preference lists of at most k voters?

The above definition regards the simplest model of bribery, where the voters are unweighted and bribing each voter requires equal effort. We also consider variants of the bribery problem where the voters have weights, where the voters have prices, and where the voters have both weights and prices.

In the weighted variant of the problem, $\mathcal{E}$-weighted-bribery, each voter $v_i \in V$ has a weight, $\omega(v_i)$. In the priced variant, $\mathcal{E}$-\$bribery, each voter $v_i \in V$ has a price, $\pi(v_i)$, for changing his or her preference list. In such a case we ask not whether we can bribe at most k people, but whether we can ensure that p is a winner by spending at most k dollars. The weighted-and-priced variant of the problem, $\mathcal{E}$-weighted-\$bribery, combines prices and weights in a natural way. For example, plurality-weighted-\$bribery problem can be described as follows:

Name:	plurality-weighted-\$bribery.
Given:	An election $E = (C, V)$ (where each voter $v_i \in V$ has a preference list over C, a nonnegative integer price $\pi(v_i)$, and a nonnegative weight $\omega(v_i)$), a distinguished candidate $p \in C$, and a nonnegative integer k (which we will sometimes refer to as *the budget*).
Question:	Is there a subcollection of voters B, such that $\pi(B) \leq k$ and there is a way to bribe the voters from B in such a way that p becomes a winner according to the plurality rule?

Manipulation is somewhat similar to bribery in that we also need to select preference lists for a group of voters, but in manipulation this group is prespecified. Formally, if $\mathcal{E}$ is some election rule then $\mathcal{E}$-manipulation is the following problem:

Name:	$\mathcal{E}$-manipulation.
Given:	An election $E = (C, V)$, where each voter has a preference list over the candidates in C, a collection S of manipulative voters (without preference lists assigned yet), and a candidate $p \in C$.
Question:	Is there a way to set preference lists of the voters in S so that, according to election system $\mathcal{E}$, p is a winner of election $(C, V \cup S)$?

The above definition is consistent with that used in the classic papers of Bartholdi, Tovey, and Trick (BTT89a) and Bartholdi and Orlin (BO91). However, we mention that sometimes one assumes that the manipulators already have some assigned preferences and that manipulation occurs only if they cast votes that do not agree with those preferences (e.g., if they switch the order of the second and the third candidate on their vote as compared to their true preferences or perform some other change that, in effect, ensures a more favorable outcome of the election for the manipulators). We also mention that since the group of manipulators may contain more than one voter, the problem is sometimes referred to as *the coalitional manipulation problem*.

Manipulation, just like bribery, can be considered in both the unweighted setting ($\mathcal{E}$-manipulation) and in the weighted one ($\mathcal{E}$-weighted-manipulation), where each voter and each manipulator has a nonnegative weight.

Let $\mathcal{E}$ be one of the election systems studied in this thesis. If it is the case that a variant of manipulation or a variant of bribery is NP-hard for $\mathcal{E}$ then we will sometimes say that $\mathcal{E}$ is resistant to this variant of bribery or manipulation. Similarly, we will sometimes say that $\mathcal{E}$ is vulnerable to a given variant of manipulation or bribery if this variant of manipulation or bribery is in P for $\mathcal{E}$. Often the notions of resistance and vulnerability are defined in a somewhat more complicated way (see, e.g., (HHR07a; FHHR07; FHHR08)), but for the case of manipulation and bribery, and our set of election systems, this definition is sufficient.

All the bribery and manipulation problems that we consider in this thesis are in the so-called constructive setting. That is, the goal of the briber and of the manipulators is to ensure that the preferred candidate is a winner. In the destructive setting we would be interested in preventing a despised candidate from being a winner.

Finally, we also mention that throughout this thesis we use the term "bribery" both in its regular sense and in the nonstandard sense of "a collection of bribes." When using the latter sense we will often speak of "a bribery," by which we will mean a collection of bribes.

3.3 Relations Between Bribery and Manipulation

For each election system $\mathcal{E}$ there are at least four variants of bribery (depending on whether the voters have prices, and whether they are weighted), and two variants of manipulation (weighted and unweighted). It will be helpful to establish some inherent relations between hardness of these variants of manipulation and bribery before we start building results for specific election systems.

Naturally, positive results (i.e., existence of polynomial-time algorithms) regarding more demanding variants of bribery and manipulation problems imply positive results about the weaker variants. For example, if weighted bribery is in P for some election system $\mathcal{E}$ then, clearly, unweighted bribery is also easy for $\mathcal{E}$. Conversely, hardness results regarding simpler models imply hardness results about the more involved ones. We will sometimes mention such implied results separately if they are interesting, but we omit them if they are not enlightening.

It is also possible to relate the complexity of bribery and the complexity of manipulation. In particular, we focus on the following two questions.

1. Is bribery always at least as hard as manipulation?
2. Is it the case that having an efficient algorithm for manipulation helps in deriving an algorithm for bribery?

The latter one can, at least to some degree, be answered in the affirmative. The reason is that to check whether a bribery of up to k voters can be successful, one can simply try all possible manipulations by k voters. In this way, for a fixed k, we can disjunctively truth-table reduce any bribery problem to the analogous manipulation problem.

Theorem 3.3. *Let k be an arbitrary positive integer. Let $\mathcal{B}$ be any of our bribery problems, but with the following constraints: Voters have no prices (i.e., we do not consider* \$bribery *problems) and bribing more than k voters is forbidden. Let $\mathcal{M}$ be the analogous manipulation problem, i.e., the manipulation problem for the same election system, with weighted voters if $\mathcal{B}$ allows that, allowing the manipulating set to contain any number of voters between* 0 *and k. Then it holds that $\mathcal{B} \leq_{\mathrm{dtt}}^{\mathrm{p}} \mathcal{M}$.*

Proof. To show that $\mathcal{B} \leq_{\mathrm{dtt}}^{\mathrm{p}} \mathcal{M}$ we need to give a polynomial-time procedure that for an input x outputs a list of strings $y_1, \ldots, y_m$ such that $x \in \mathcal{B}$ if and only if at least one of y_i, $1 \leq i \leq m$, is in $\mathcal{M}$. We now describe such a procedure.

Let x be the input string. We first check whether x can be parsed as an instance of $\mathcal{B}$ (reminder: that is, that x meets the syntactic constraints of $\mathcal{B}$). If not then we output an empty list and terminate; otherwise we decode election $E = (C, V)$, the preferred candidate p, and $k' \leq k$, the maximum number of bribes we can use, from the string x. For every subset W of at most k' elements (we say "at most k'" rather than "exactly k'" simply because of the possibility that $k' \geq \|V\|$; one could alternatively focus simply on "exactly $\min(k', \|V\|)$") of V we form an instance of the manipulation problem with election $(C, V - W)$, preferred candidate p, and the manipulator collection W. After we go through all at-most-k'-element subsets we output the list of all the manipulation instances that we formed.

This procedure clearly works in polynomial time as there are at most $\binom{\|V\|}{k} = O(\|V\|^k)$ sets to test and we can form instances of manipulation in polynomial time. If any of the manipulation instances we output is in $\mathcal{M}$ then bribery is possible: It is enough to bribe exactly the voters selected for the manipulating group. On the other hand, if bribery is possible, then at least one of the instances we output belongs to $\mathcal{M}$: namely any one that includes the voters we would bribe. ❑

While simple, this result is still powerful enough to allow the inheritance of some results from previous papers. Bartholdi, Tovey, and Trick (BTT89a) discuss manipulation by single voters and Theorem 3.3 translates their results to the bribery case. In particular, this translation says that bribery for $k = 1$ is in P for plurality, Borda count, and many other systems.

Can we strengthen Theorem 3.3 from constant-bounded bribery to general unweighted bribery? The answer is no: There are election systems for which bribery is NP-complete but manipulation is in P. In particular, manipulation for approval voting (both in the weighted and the unweighted case) is in P for any size of manipulating set: The manipulating group approves only its favorite candidate and nobody else. However, by Theorem 4.15, bribery for approval voting is NP-complete. (Although, of course, when the number of bribes is bounded by some fixed constant then, by Theorem 3.3, approval-bribery can be solved in polynomial time.)

Let us now turn to our first question: Is bribery always at least as hard as manipulation? In Theorem 3.3 we managed to disjunctively truth-table reduce a restricted version of bribery to manipulation. Is it possible to reduce manipulation to bribery? At first, this appears to be more difficult because bribery allows more freedom to the person interested in affecting the elections, and to embed manipulation within bribery we would have to find some way of expressing the fact that only a certain group of voters should be bribed (or, at least, expressing the fact that if there is *any* successful bribery then there is also one that only bribes the manipulators). However, this is in fact easy, provided that we are willing to pay the price of reducing to \$bribery rather than to an unpriced variant of bribery.

Theorem 3.4. *Let $\mathcal{M}$ be some manipulation problem and let $\mathcal{B}$ be the analogous* \$bribery *problem (for the same election system). It holds that $\mathcal{M} \leq_{\rm m}^{\rm p} \mathcal{B}$.*

Proof. Let M be an instance of $\mathcal{M}$ containing an election $E = (C, V)$, manipulator collection S, and a preferred candidate $p \in C$. We form an instance B of $\mathcal{B}$ such that B contains preferred candidate p, budget 0, and election $E' = (C, V' \cup S')$, where:

1. V' is equal to V, except that each voter has price 1, and
2. S' is equal to S, except that each voter has price 0 and some fixed arbitrary preference list.

Since the bribery budget is set to zero, the only voters that we may possibly bribe are those in S'. The preference lists of the voters in S' after any such bribery directly correspond to a manipulation in M. This reduction can be carried out in polynomial time. ❑

Naturally, Theorem 3.4 holds even for \$bribery problems where prices are represented in unary or are required to come from the set $\{0, 1\}$.

Unfortunately, in general Theorem 3.4 cannot be strengthened to not require reducing to a priced variant of bribery (unless P = NP, but if P = NP then there is no need for such a reduction). We show an artificial election system where bribery is in P but manipulation is NP-complete.

Theorem 3.5. *There exists a voting system $\mathcal{E}$ for which manipulation is* NP*-complete, but bribery is in* P*.*

Proof. By $\langle \cdot, \cdot \rangle$ we mean a fixed, natural pairing function for strings, i.e., one that is computable in polynomial time, has polynomial-time computable projection functions, and that satisfies $|\langle x, y \rangle| = 2(|x| + |y| + 1)$.

Let A be an NP-complete set and let $B \in \mathrm{P}$ be such that

1. $A = \{x \in \Sigma^* \mid (\exists y \in \Sigma^*)[\langle x, y \rangle \in B]\}$, and

2. $(\forall x, y \in \Sigma^*)[\langle x, y\rangle \in B \Rightarrow |x| = |y|]$.

Such a set B can easily be constructed from any NP-complete set by padding. The idea of the proof is to embed a verifier for A within the election rule $\mathcal{E}$. We do this in a way that forces manipulation to solve arbitrary A instances, while allowing bribery to still be easy.

First, we observe that preference lists can be used to encode arbitrary binary strings. We will use the following encoding. For C a set of candidates, let $c_1, c_2, \ldots, c_m$ be those candidates in lexicographical order. We will view the preference list

$$c_{i_1} > c_{i_2} > c_{i_3} > \cdots > c_{i_m}$$

as an encoding of the binary string $b_1 b_2 \cdots b_{\lfloor m/2 \rfloor}$, where

$$b_j = \begin{cases} 0 & \text{if } i_{2j-1} > i_{2j}, \\ 1 & \text{otherwise.} \end{cases}$$

This encoding is of course not the most efficient one, and a given binary string may have many preference lists that encode it. However, this encoding is very easy and has the properties that we need in our construction.

In our reduction, binary strings starting with 1 will encode instances (i.e., strings over Σ, whose membership in A we want to test), and binary strings starting with 0 will encode witnesses (in our case, a string $y \in \Sigma^*$ is a witness for a string $x \in \Sigma^*$ if $\langle x, y\rangle$ is in B). Given this setup, we can describe our election system $\mathcal{E}$. Let (C, V) be an election. For each $c \in C$, c is a winner of the election if and only if $\|V\| = 3$ and

Rule 1: all preference lists encode strings starting with 1 or all preference lists encode strings starting with 0, or

Rule 2: exactly one preference list encodes a string that starts with 1, say $1x$, and at least one other preference list encodes a string $0y$ such that $\langle x, y\rangle \in B$.

Thus, either all candidates are winners or none of them are winners. Note that testing whether a candidate c is a winner of an $\mathcal{E}$ election can easily be done in polynomial time.

The following polynomial-time algorithm shows how to perform an optimal bribery. This implies that $\mathcal{E}$-bribery $\in$ P.

1. If c is a winner, we do nothing.

2. Otherwise, if $\|V\| \neq 3$, then bribery is impossible.

3. Otherwise, if there is exactly one voter whose preference list encodes a string that starts with 1, then we bribe that voter to encode a string that starts with 0. By Rule 1, c is a winner of the election.

4. Otherwise, there is exactly one voter whose preference list encodes a string that starts with 0 and we bribe that voter so that his or her preference list encodes a string that starts with 1. By Rule 1, c is a winner of the election.

On the other hand, the ability to solve the manipulation problem for $\mathcal{E}$ implies the ability to solve A. We construct a reduction from A to $\mathcal{E}$-manipulation. Given a string $x \in \Sigma^*$, we first check whether $\langle x, 0^{|x|}\rangle \in B$. If so, then clearly $x \in A$ and we output some fixed member of $\mathcal{E}$-manipulation. Otherwise, we output a manipulation problem with candidates $\{1, 2, \ldots, 2(|x|+1)\}$ and three voters, v_0, v_1, and v_2, such that v_0's preference list encodes $1x$, v_1's preference list encodes $00^{|x|}$, and v_2 is the only manipulative voter. We claim that candidate 1 can be made a winner if and only if $x \in A$.

Since $\langle x, 0^{|x|}\rangle \notin B$, the only way in which v_2 can make 1 a winner is when v_2 encodes a string $0y$ such that $\langle x, y\rangle \in B$ in which case $x \in A$. For the converse, if $x \in A$, there exists a string $y \in \Sigma^{|x|}$ such that $\langle x, y\rangle \in B$. We can encode string $0y$ as a preference list over $\{1, 2, \ldots, 2(|x|+1)\}$, and let this be the preference list for v_2. This ensures that 1 is a winner of the election.

Since this reduction can be computed in polynomial time, and the $\mathcal{E}$-manipulation's membership in NP is clear, we have that $\mathcal{E}$-manipulation is NP-complete. ❑

The above election system is very unnatural. For example, depending on the votes, its winner set is either the set of all candidates or an empty set. With some work one could modify the above proof to use a slightly more realistic system, but this would make the proof more involved and the system would not become practically useful anyway. The main implication of Theorem 3.5 is that unless we somehow restrict our election rules or prove P = NP, obtaining a general reduction from manipulation to bribery seems precluded. It would be interesting to find a natural election system that also has the property that manipulation is difficult but bribery is easy. Alternatively, it is an interesting research direction to seek a natural social-choice-theoretic condition such that if a given election system satisfies this condition then its manipulation problem reduces to its bribery problem.

4 Plurality and Utility-Based Systems

In this chapter we study the complexity of bribery in plurality voting and the complexity of nonuniform bribery in utility-based voting. Plurality rule is perhaps the most popular election system in practical use. It is simple to understand and from the point of view of democracy it is very natural and appealing to make a decision that many people prefer. However, plurality is not perfect. For example, plurality rule may slight the voices of minorities and does not take into account full information about voters' preferences. In particular, if there is some candidate that all voters rank as second best and no other candidate is the top choice of many rankings, it might seem natural to elect this "second best" person. Unfortunately, plurality is blind to this situation. Despite its limitations, widespread use of plurality makes the results of this chapter particularly relevant.

Let us briefly restate how plurality and utility-based voting work, and explain why studying them jointly is natural. Let $E = (C, V)$ be an election. Under plurality each voter assigns one point to his or her favorite candidate. In the bribery problem for plurality our goal is to find a group of voters who we can afford to bribe, and bribing whom ensures that our preferred candidate is a winner. While the task of selecting the group of voters to bribe is not necessarily trivial, bribery in plurality is greatly simplified by the fact that each voter has only a single point to assign. The briber does not need to worry about any correlations between the votes, which can possibly become very complicated. To see that this is a major simplification, consider the setting in which the voters distribute several points each and, for some reason, the briber has to take away at least one point from each candidate in some subset C' of the candidates. This means the briber, in essence, has to find a minimum set of voters whose votes "cover" the candidates in C'. This by itself is a difficult problem (in fact, we use its difficulty to show that bribery for approval voting is NP-complete; Theorem 4.15), let alone the fact that reassigning the points of the bribed voters would also require care.

In utility-based voting (see Definition 3.2) each voter does have some number of integral points to distribute among the candidates and, as in plurality, the candidates with most points are the winners. In addition, we often impose an upper bound on the number of points a voter can assign to a single candidate, and sometimes we allow the voters not to use up all of their points. Examples of systems that can be represented as utility-based voting include approval, k-approval, veto, and, of course, plurality.

As we have just argued, bribery in utility-based voting can be much harder than in plurality. However, this hardness seems to come from the fact that, in our standard model of bribery, bribing a voter means obtaining full control over his or her vote. While in some situations this is a reasonable model, in others it may be more appropriate to consider a more refined price model. For example, a voter might be perfectly happy to swap two neighboring candidates on his or her preference list, but might outright refuse to swap his or her most preferred candidate with his or her most despised one. In Section 4.2 we study a variant of bribery in utility-based voting where we pay each voter separately for each of his or her points that we reassign. We call this model *nonuniform bribery*.

Now it becomes apparent why it is natural to consider bribery in plurality and nonuniform bribery in utility-based systems jointly. Because of its flexible pricing, nonuniform bribery in utility-based systems is very similar to regular bribery in plurality. In nonuniform bribery, we can reassign each point cast by each voter independently, analogously to our regular bribery in plurality. Thus, many results in Section 4.2 are inspired by the techniques developed for bribery in plurality and, since plurality itself is an incarnation of utility-based voting, many of the results regarding nonuniform bribery apply to plurality.

This chapter is devoted to the study of bribery, but we mention that many manipulation results can be directly obtained via applying translation techniques from

Section 3.3.

4.1 Bribery in Plurality

We will view a vote in plurality rule elections as a vote for a particular candidate, namely, the most preferred candidate according to the preference list that is the actual vote (for the case of bribery this is the only thing that matters about the voter—although in other contexts, such as control via deletion of candidates (BTT92; HHR07a; HHR07b), the full ordering might be important).

The simplest bribery scenario is when the voters are unweighted and bribing each voter requires equal effort. Not surprisingly, bribery is easy in this setting.

Theorem 4.1. plurality-bribery *is in* P*.*

Proof. The proof of this theorem is simple, but we describe it in detail as a simple introduction to our proofs regarding bribery. We will give a polynomial-time algorithm that given a plurality election $E = (C, V)$, a preferred candidate $p \in C$, and a budget k, decides whether it is possible to make p a winner by bribing at most k voters.

Our algorithm works in the following way. Initially we have bribed zero voters. We check whether p currently is a winner. If so, we accept. Otherwise, until doing so will exceed the bribe limit, we pick any current winner, bribe one of his or her voters to vote for p, and jump back to the testing whether p is a winner. If we reach the bribe limit (i.e., in the above we have the "until doing so will exceed the bribe limit" break us out of the loop) without making p a winner then we reject.

If this algorithm accepts then obviously bribery is possible. We now show that if it is possible to ensure that p is a winner via at most k bribes then our algorithm accepts. Our proof follows by induction on k. For the base case it is enough to note that the algorithm works correctly for $k = 0$. For the induction step, let us assume that for each election $E' = (C', V')$, $p \in C$, and budget k' such that $k' < k$, where k is some positive integer, the algorithm accepts exactly if it is possible to ensure that p is a winner via at most k' bribes. Now, consider an arbitrary input $E = (C, V)$, $p \in C$, and budget k, such that p can become a winner via at most k bribes. We will show that our algorithm accepts this input. We consider two cases. If there is a bribery of up to k voters that ensures p's victory but that never involves any of the current winners of election (C, V) then it is clear that our algorithm accepts. (Let $V_p \subseteq V$ be the set of all voters who do not vote for p. In this case any bribery of $\min(k, ||V_p||)$ voters ensures that p becomes a winner.) Thus, let us assume that all briberies of k voters that make p a winner involve bribing at least one of the current winners. Let $E'' = (C, V'')$ be an election obtained from E via bribing one of the winners of (C, V), call him c_1. When we run our algorithm on input E, p, k then in the first iteration the algorithm bribes one of the winners of (C, V) and obtains an instance of plurality-bribery that is isomorphic to $E'', p, k-1$. Then, in the further iterations the algorithm behaves the same as if it got an input instance isomorphic to $E'', p, k-1$ and thus, by our inductive hypothesis, accepts.

The algorithm works in polynomial time because $\|V\|$ bribes suffice to make p a winner and each of the iterations can be executed in polynomial time. The proof is complete. ❑

The ease of obtaining the above algorithm might fool us into thinking that bribery within the plurality system is always easy. However, that is not so.

Theorem 4.2. plurality-weighted-\$bribery *is* NP*-complete, even for just two candidates.*

Proof. plurality-weighted-\$bribery is in NP: We can guess the voters to bribe and test whether such a bribe both makes our designated candidate a winner and does not exceed the budget. It remains to show that the problem is NP-hard.

To show NP-hardness, we give a reduction from Partition. Let $s_1, \ldots, s_n$ be a sequence of nonnegative integers, an input for Partition, and let $\sum_{i=1}^{n} s_i = 2S$. We form an election with two candidates, p and c, and exactly n voters, $v_1, \ldots, v_n$, with each v_i having both weight and price equal to s_i. All voters prefer c to p. The budget k is set to S. We claim that p can become a winner if and only $s_1, \ldots, s_n$ can be partitioned into two equal-sum groups.

Let us assume that there is a set $A \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in A} s_i = S$. Clearly, bribing the voters in $\{v_i \mid i \in A\}$ to vote for p ensures that p is a winner and costs exactly S. On the other hand, assume that p can be made a winner by bribes of total cost at most $k = S$. The weight of each voter is equal to his or her price and so p can obtain at most vote weight $k = S$. In fact, p must obtain exactly vote weight S, since from our setup it is clear that if p gains strictly less than vote weight S then c will be the unique winner. This means that there is a way of picking some voters whose weights sum up to exactly S, and thus the sequence $s_1, \ldots, s_n$ can be partitioned into two subsequences that each sum up to S. Our reduction can be carried out in polynomial time and so the proof is complete. ❑

The above theorems show that bribery is easy in the basic case but becomes intractable if we allow for voters with prices and weights. It is natural to ask which of the additional features (prices? weights?) is responsible for making the problem difficult. It turns out that neither of them is the sole reason and that only their combination yields enough power to make the problem NP-complete.[1]

Theorem 4.3. *Both* plurality-\$bribery *and* plurality-weighted-bribery *are in* P*.*

Theorem 4.3 is a special case of a result that we prove later (namely, of Theorem 4.6) and thus, instead of giving the proof, we provide an informal discussion of how one could obtain algorithms for plurality-\$bribery and for plurality-weighted-bribery.

A direct greedy algorithm, like the one underpinning Theorem 4.1, fails to prove Theorem 4.3: The problem is that one has to judge whether it is better to bribe the voters who currently prefer one of the winners, or if it is better to bribe the voters with the highest weights (or lowest prices). To see that the former may sometime make sense, consider an election in which a has two weight-4 voters, b has one weight-5 voter, and p has one weight-2 voter. Bribing one weight-4 voter is a winning bribery but bribing the one weight-5 voter is not.

[1]However, it is interesting to compare this to Theorems 5.2, 5.3, 5.7, and 5.8, which suggest that in general it is often high weights that are responsible for making bribery problems NP-complete.

However, we may approach Theorem 4.3's proof as follows. Assume that p will have at least r votes after the bribery (or, in the weighted case, r vote weight), where r is some number to be specified later. If this is to make p a winner, we need to make sure that everyone else gets *at most* r votes. Thus we carefully choose enough cheapest (heaviest) voters of candidates that defeat p so that after bribing them to vote for p each candidate other than p has at most r votes. Then we simply have to make sure that p gets at least r votes by bribing the cheapest (the heaviest) of the remaining voters. If during this process p ever becomes a winner without exceeding the budget (the bribe limit) then we know that bribery is possible.

However, how do we pick the value of r? In the case of plurality-\$bribery, we can simply run the above procedure for all $\|V\|$ possible values, and accept exactly if it succeeds for at least one of them. For plurality-weighted-bribery a slightly trickier approach works. Namely, it is enough to try all values r that can be obtained as a vote weight of some candidate (other than p) via bribing some number of his or her heaviest voters. There are only polynomially many such values and so the whole algorithm works in polynomial time. The intuition for using such values r is the following: (a) When bribing voters of some candidate one can always limit oneself to the heaviest ones, and (b) after each successful bribery there is a value r' such that p's vote weight is at least r', each other candidate's vote weight is at most r', and there is some candidate $c \neq p$ such that c's vote weight is exactly r'. Our algorithm, in essence, performs an exhaustive search (within our heavily limited search space) for such a value r'.

Theorems 4.2 and 4.3 state that plurality-weighted-\$bribery is NP-complete but any attempt to make it simpler immediately pushes it back to the realm of P. In fact, the situation is even more dramatic. In the NP-complete problem plurality-weighted-\$bribery we assume that both prices and weights are encoded in binary. However, if either the prices or the weights are encoded in unary, then the problem, again, becomes easy.

Before we proceed with a formal proof of this fact, let us discuss the issue in an informal manner. Why does the unary encoding of the weights or of the prices matter? The reason is that, for example, if the weights are encoded in unary then there trivially are only linearly many (with respect to the size of the input problem) different total weights of subsets of voters. Together with some additional tricks this allows us to use dynamic programming to obtain a solution.

It is tempting to use exactly the same proof approach as the one that we hinted on in the discussion below Theorem 4.3, that is, to split the bribery into two parts: demoting others and promoting p. However, doing so would not be correct. Sometimes the optimal way of getting the scores of other candidates to be at most at a certain threshold r prevents one from getting an optimal bribe for the complete problem. Consider elections with two candidates, c and p, and two voters v_1 and v_2 such that v_1 has both price and weight equal to 10, and v_2 has both price and weight equal to 7. Both v_1 and v_2 prefer c to p. The optimal way of getting c down to vote weight at most 10 is by bribing v_2. However, at that point making p a winner requires bribing v_1 as well. Yet, bribing just v_1 is a cheaper way of making p a winner and getting c below the 10 threshold.

Definition 4.4. plurality-weighted-\$bribery$_{\text{unary}}$ *is defined exactly as is* plurality-weighted-\$bribery*, except the prices are to be encoded in unary.* plurality-weighted$_{\text{unary}}$-\$bribery *is* plurality-weighted-\$bribery *except with the weights encoded in unary.*

We will refer to plurality-weighted-\$bribery$_{\text{unary}}$ as the "unary prices case," and to plurality-weighted$_{\text{unary}}$-\$bribery as the "unary weights case." We will now give an overview of how the algorithm works in the unary prices case, when given an election $E = (C, V)$, preferred candidate $p \in C$, and budget k. The unary weights case can be handled analogously. The main idea is that, using the fact that there are only linearly many possible prices to be paid, we can argue that there exists a polynomial-time computable function $\mathit{Heaviest}(E, C', \pi, r)$—where C' will be a subset of the candidates, π will be an integer price, and r will be an integer threshold—that gives the maximum vote weight that we can obtain by bribing voters of candidates in C' such that

1. the cost of this bribery is at most π,

2. after the bribery every candidate in C' has vote weight at most r.

To test whether it is possible to make p a winner by spending at most k dollars, we need to check if there is a threshold r such that $\mathit{score}_E(p)+\mathit{Heaviest}(E, C-\{p\}, k, r) \geq r$. Unfortunately, in the case of plurality-weighted-\$bribery$_{\text{unary}}$ we cannot just try all thresholds since there may be exponentially many of them. Instead we use a strategy similar to the one that we hinted on when discussing Theorem 4.3. After every successful bribery (in elections with at least two candidates) there is some candidate $c \neq p$—namely, the candidate(s) other than p with the greatest post-bribery total weight—that either is a tied-with-p winner or loses only to p. We can use the after-bribery vote weight of this candidate to be the threshold for the bribery of the voters of all the other candidates. Of course, we neither know who this candidate is nor what vote weight he or she would have after a successful bribery. Nonetheless, we can try all candidates $c \neq p$ and for each such candidate and each possible "sub-budget" $b \leq k$ we can ask what is the maximum amount of additional weight we can get for p from bribing c's voters when allowed to spend at most b to do so. Then, using the thus obtained threshold, we can bribe the voters of the rest of the candidates. There are (at most) linearly many candidates and (at most) linearly many prices so this yields (at most) polynomially many combinations.

Let us now describe how the above plan can be implemented. We no longer limit ourselves to the unary prices case, but describe both cases in parallel. Let $E = (C, V)$, p, and k be our input. For each candidate $c \in C$ we define

$$V_E^c = \{v \in V \mid v\text{'s most preferred candidate is } c\}.$$

Naturally, we ask each voter that we choose to bribe to vote for p. For a given candidate $c \in C$, we can describe our bribing options either as a function that gives the highest weight of c's voters we can bribe for b dollars or as a function that gives the lowest price needed to gain vote weight at least w by bribing c's voters.[2]

$$\begin{aligned} \mathit{heaviest}(c, b) &= \max\{\omega(U) \mid U \subseteq V_E^c \text{ and } \pi(U) \leq b\}. \\ \mathit{cheapest}(c, w) &= \min\{\pi(U) \mid U \subseteq V_E^c \text{ and } \omega(U) \geq w\}. \end{aligned}$$

[2]We assume that election E is an implicit argument for these functions. The same applies to functions *Heaviest* and *Cheapest* defined later in this discussion.

If c is not a candidate in C, these functions are undefined. Here and in the rest of the proof, we take the max and min of the empty set to be undefined. Note that if c is a candidate in E, then $\mathit{heaviest}(c, b)$ is defined for all $b \geq 0$ and $\mathit{cheapest}(c, w)$ is defined for all $w \leq \omega(V_E^c)$. Also note that *heaviest* can easily be computed in polynomial time in the unary prices case and that *cheapest* can easily be computed in polynomial time in the unary weights case. In both cases we simply use dynamic programming solutions for the knapsack problem.[3] We can further generalize these functions to give us information about the best bribes regarding sets of candidates. For a subset U of voters, by $\mathit{bribed}(E, U)$ we mean election E with voters in U bribed to vote for p. We define

$$\begin{aligned}
\mathit{Heaviest}(C', b, r) &= \max\left\{ \omega(U) \;\middle|\; \begin{array}{l} (U \subseteq \bigcup_{c \in C'} V_E^c) \wedge (\pi(U) \leq b) \wedge \\ (\forall c \in C')[\mathit{score}_{\mathit{bribed}(E,U)}(c) \leq r] \end{array} \right\}, \text{ and} \\
\mathit{Cheapest}(C', w, r) &= \min\left\{ \pi(U) \;\middle|\; \begin{array}{l} (U \subseteq \bigcup_{c \in C'} V_E^c) \wedge (\omega(U) \geq w) \wedge \\ (\forall c \in C')[\mathit{score}_{\mathit{bribed}(E,U)}(c) \leq r] \end{array} \right\}.
\end{aligned}$$

If C' is not a subset of E's candidate set, these functions are undefined.

Lemma 4.5. *We consider now only elections in which each voter has both a price and a weight. If prices are encoded in unary then there is an algorithm that computes Heaviest in polynomial time. If weights are encoded in unary then there is an algorithm that computes Cheapest in polynomial time.*

Proof. Note that in the unary prices case there are only linearly many sub-budgets b for which we need to compute the value of *Heaviest*, namely $0 \leq b \leq \pi(V)$, and in the unary weights case there are only linearly many weights w for which we need to evaluate *Cheapest*, namely $0 \leq w \leq \omega(V)$. Using this fact we provide dynamic programming

[3]The knapsack problem is the following. Given a set of items, each with a price and a weight, is it possible to select items with total weight at least W, but without exceeding total price K? It is well known that the knapsack problem has a polynomial-time dynamic programming algorithm if either the prices are encoded in unary or the weights are encoded in unary. (See (MT90) for background/reference on the knapsack problem.)

algorithms for computing both functions. For the base case we have the following: If c is not a candidate of E, then both our functions are undefined. Otherwise,

$$\mathit{Heaviest}(\{c\}, b, r) = \begin{cases} \mathit{heaviest}(c, b) & \text{if } \mathit{score}_E(c) - \mathit{heaviest}(c, b) \leq r, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

$$\mathit{Cheapest}(\{c\}, w, r) = \begin{cases} \mathit{cheapest}(c, w) & \text{if } \mathit{score}_E(c) - w \leq r, \\ \mathit{cheapest}(c, \mathit{score}_E(c) - r) & \text{otherwise.} \end{cases}$$

The following observation allows us to compute *Cheapest* and *Heaviest* for larger sets. We assume that C' does not contain c. If any of the candidates in $C' \cup \{c\}$ are not candidates of E, then both our functions are undefined. Otherwise,

$$\mathit{Heaviest}(C' \cup \{c\}, b, r) = \max\{\mathit{Heaviest}(C', b', r) + \mathit{Heaviest}(\{c\}, b - b', r) \mid$$
$$0 \leq b' \leq b \text{ and } \mathit{Heaviest}(C', b', r) \text{ and } \mathit{Heaviest}(\{c\}, b - b', r) \text{ are both defined}\}.$$
$$\mathit{Cheapest}(C' \cup \{c\}, w, r) = \min\{\mathit{Cheapest}(C', w', r) + \mathit{Cheapest}(\{c\}, w - w', r) \mid$$
$$0 \leq w' \leq w \text{ and } \mathit{Cheapest}(C', w', r) \text{ and } \mathit{Cheapest}(\{c\}, w - w', r) \text{ are both defined}\}.$$

Thus, in the unary prices case we can compute $\mathit{Heaviest}(C', b, r)$ using dynamic programming in polynomial time. The same applies to $\mathit{Cheapest}(E, C', w, r)$ for the unary weights case. ❏

Theorem 4.6. *Both* plurality-weighted-\$bribery$_{\text{unary}}$ *and* plurality-weighted$_{\text{unary}}$-\$bribery *are in* P.

Proof. Algorithms for both of the problems are very similar and we will describe only the unary prices case in detail. We provide the pseudocode for the unary weights case, but we omit its proof of correctness, which is analogous to the proof for the unary prices case.

Figure 4.1 shows our procedure for the unary prices case. The idea of the algorithm is the following: Suppose that there is a set B of voters such that if we bribe all members of B to vote for p then p becomes a winner. We can assume that for each candidate c, c's voters have been bribed optimally, i.e., there is no cheaper way of getting the same (or greater) vote weight by bribing a different subset of c's voters. There is some

```
procedure $UnaryPricesBribery(C, V, p, k)$
begin
    $C' = C - \{p\}$;
    if $k \geq \pi(V)$ then accept;
    for $c \in C'$ do
        for $b$ such that $0 \leq b \leq k$ do
        begin
            $w' = heaviest(c, b)$;
            $r = score_E(c) - w'$;
            $w = Heaviest(C' - \{c\}, k - b, r)$;
            if $w$ is defined and $score_E(p) + w + w' \geq r$ then accept;
        end
    reject;
end
```

Figure 4.1: The main procedure for plurality-weighted-\$bribery$_{\text{unary}}$.

candidate c that has the most votes among the non-p candidates after bribery. Thus, to decide if bribery is possible it is enough to test whether there is a candidate $c \neq p$ and a sub-budget b, $0 \leq b \leq k$, such that after bribing c's voters optimally, spending b dollars, it is still possible to bribe (without overall exceeding the budget) voters of the other candidates in such a way that

1. each candidate ends up with vote weight not higher than that of c, and
2. enough voters can be bribed so that p becomes a winner.

Our algorithm tests exactly if this is the case and accepts if so. (Though its "if-then" line might at first seem to focus just on having the candidates in $C - \{c\}$ beat p, note that c's post-bribery score is r, so that line handles c also.) By the above reasoning, if bribery is possible the algorithm accepts. It should also be clear that if the algorithm accepts then bribery is indeed possible. Since the functions *heaviest* and *Heaviest* can

```
procedure $UnaryWeightsBribery(C, V, p, k)$
begin
    $C' = C - \{p\}$;
    for $c \in C'$ do
        for $w'$ such that $0 \leq w' \leq \omega(V_E^c)$ do
        begin
            $b = cheapest(c, w')$;
            $r = score_E(c) - w'$;
            $b' = Cheapest(C' - \{c\}, r - (score_E(p) + w'), r)$;
            if $b'$ is defined and $b + b' \leq k$ then accept;
        end
    reject;
end
```

Figure 4.2: The main procedure for plurality-weighted$_{\text{unary}}$-\$bribery.

be computed in polynomial time, we have that the whole algorithm runs in polynomial time. Thus, plurality-weighted-\$bribery$_{\text{unary}}$ is in P.

An analogous algorithm works for the unary weights case, see Figure 4.2. The proof of correctness is analogous to the unary prices case. ❑

Theorem 4.6 is particularly interesting because, together with Theorem 4.2, it says that plurality-weighted-\$bribery is difficult only if we choose both weights and prices to be high. However, the prices are set by the voters and, in many cases, one would expect that they would set them to be fairly low, rendering the bribery problem easy. Alternatively, the briber may him or herself rescale the prices, obtaining a near-perfect solution without the need to solve an NP-complete problem. The next theorem formalizes this approach.

Theorem 4.7. *There is an FPTAS for the problem of finding a minimum-cost successful bribery in a weighted plurality election where the voters have price tags.*

Proof. Our input instance I contains a plurality election $E = (C, V)$ where voters are weighted and have price tags, a preferred candidate $p \in C$, and a positive rational value ε, $0 < \varepsilon < 1$. By $\pi_{\max}$ we mean the highest price occurring in I and we assume that $\pi_{\max}$ is at least 1. Otherwise the solution where *every* voter is bribed to vote for p would be optimal. We let n be the number of voters. Thus, $n\pi_{\max}$ is an upper bound on the cost of any bribery within I.

The high-level idea of the algorithm is to scale down the prices so that they are polynomially bounded in n and $\frac{1}{\varepsilon}$, and to run the polynomial-time algorithm for plurality-weighted-\$bribery$_{\text{unary}}$ from Theorem 4.6. However, I might include some voters with very high prices that are not needed in an optimal solution. By using a scaling factor appropriate for these high prices we may lose all the information regarding the smaller, useful, prices. Thus, instead of performing one scaling, we perform polynomially many of them, starting with small factors and going toward the larger ones.

Our algorithm executes $\lceil \log \pi_{\max} \rceil$ iterations. In each iteration variable t contains our current guess of an upper bound on the largest price used within the optimal solution. We start with $t = 1$ and we double it after every iteration.

Given a value of t, an iteration is executed as follows. Set $K = \frac{t\varepsilon}{n}$ and construct an instance I' that is identical to I, only that: (a) each price q such that $q \leq t$, is replaced by $\lceil \frac{q}{K} \rceil$ (note that $\lceil \frac{q}{K} \rceil \leq \lceil \frac{n}{\varepsilon} \rceil$), and (b) any price higher than t is replaced by $\frac{1+2\varepsilon}{\varepsilon} n^2 + 1$. Due to this transformation, each price in I' is polynomially bounded in $\frac{1}{\varepsilon}$ and n. We find an optimal solution for I' using the polynomial-time algorithm for plurality-weighted-\$bribery$_{\text{unary}}$ from the proof of Theorem 4.6. If the solution has cost $\frac{1+2\varepsilon}{\varepsilon} n^2 + 1$ or higher then we discard it and otherwise we store it for future use otherwise.

When all the iterations are finished, we return a stored solution with the lowest cost. There is at least one stored solution because in the final iteration we have $t \geq \pi_{\max}$ and so the final iteration does not discard its solution.

We claim that this algorithm finds a solution whose cost is within 2ε of the optimal one. Let O be an optimal solution and let $u = \max\{\pi(v_i) \mid v_i \text{ is bribed within } O\}$. Let us consider an iteration of our algorithm with t such that $\frac{t}{2} \leq u \leq t$. Let S be our optimal solution to instance I' in this iteration. Since I and I' differ only in voter's prices, either of S and O is a valid solution for either of I and I'. By $\pi(S)$ and $\pi(O)$ we mean the costs of briberies specified in S and in O, respectively, expressed using prices from instance I. By $\pi'(S)$ and $\pi'(O)$ we mean analogous values, but with respect to prices in I'. There is a solution to I' of cost less than $\frac{1+2\varepsilon}{\varepsilon}n^2 + 1$. To see this, it is enough to note that O involves bribing at most n voters, that in I' each of these voters has price at most $\lceil \frac{t}{K} \rceil$, and that $n\lceil \frac{t}{K} \rceil \leq n\lceil \frac{n}{\varepsilon} \rceil \leq \frac{n^2+n\varepsilon}{\varepsilon} < \frac{1+2\varepsilon}{\varepsilon}n^2 + 1$. S is an optimal solution for I' so $\pi'(S) < \frac{1+2\varepsilon}{\varepsilon}n^2 + 1$ and thus S is not discarded. It holds that

$$\pi(O) \leq \pi(S) \leq K\pi'(S) \leq K\pi'(O).$$

The first inequality holds because O is an optimal solution to I. The second one follows because instance I' has prices rounded up. The last inequality is due to the fact that S is an optimal solution for I'.

Since $\frac{t}{2} \leq u \leq t$ and S is not discarded, for each price q of I of a voter bribed in either O or S we have a corresponding price $q' = \lceil \frac{q}{K} \rceil$ in I'. Due to rounding, it is easy to see that for each such q and q', it holds that $q \leq Kq' \leq q + K$. Since any bribery involves bribing at most n voters we have that

$$K\pi'(O) \leq \pi(O) + nK.$$

Since $nK = \varepsilon t$, and in this iteration we have $\frac{t}{2} \leq u \leq t$, we have that $\frac{t}{2} \leq u \leq \pi(O)$. Thus, the following sequence of inequalities holds.

$$\pi(S) \leq K\pi'(S) \leq \pi(O) + \varepsilon t \leq \pi(O) + 2\varepsilon\pi(O) = (1 + 2\varepsilon)\pi(O).$$

Since S is not discarded and the other iterations may only improve the solution output, the final solution the algorithm outputs is within 2ε of the optimal one. The proof is complete. ❏

4.2 Nonuniform Bribery

In the previous section we assumed that each voter has a single price irrespective of how the briber modifies the vote's preference list. This model is often useful, but there are scenarios in which a more flexible pricing scheme is more natural. For example, consider an election with three candidates, a, b, and c, and a particular voter who prefers a to b to c, but who actually likes both a and b and who absolutely hates c. Such a voter may be willing, at a small price, to change his or her vote to rank b first, but would never, regardless of the bribe, change the vote to rank c first.

A different example where it is useful to model voters as having such nonuniform prices is best seen from the point of view of the briber. A briber that wants some candidate p to win might want to follow a certain *policy* in his or her bribing. For example, he or she might not want to bribe anyone to vote for p in order not to cast "bad light" on p. Such a briber would have to make p a winner via bribes that redistribute other candidates' support. Using the nonuniform model of bribery one could express this policy via setting the prices for moving points to p so high as to be outside of the allowed budget. We can easily come up with other meaningful policies.

Yet another scenario where nonuniform bribery model is useful regards the issue of coalition formation. Consider an election where one of the voters realizes that his or her option is very unlikely to win, but where there are many other voters that support options similar, but slightly different, from his or hers. Such a voter might want to find out which of the others, but as few as possible, he or she would have to convince to form a coalition with him or her in order to have enough voting power to choose an option that all of them would be reasonably satisfied with. One way to compute such a set would be to: (a) find a group of voters that currently vote for options similar to the agent's, (b) form nonuniform bribery instance where those voters can be bribed, at a relatively low price, to vote for the agent's option and where all other briberies are either very expensive or impossible (beyond budget), (c) compute a minimum-cost nonuniform bribery that ensures that the agent's favorite option wins. The voters involved in this bribery would be candidates for the coalition.

Thus, the issue of nonuniform bribery is both important and useful, even in the cases where we are not really "bribing" anyone, but simply are trying to strategically plan our behavior. In this section we give a number of results that show that the problem of nonuniform bribery can often be solved in polynomial time in the case of unweighted voters, but often becomes NP-complete if the voters are weighted, in some cases even if the values of price functions are encoded in unary.

Throughout this section we use the utility-based voter model instead of our standard rational voter model.

4.2.1 Nonuniform Bribery Model

Let e and b be two integer values. We define the (e, b)-bribery problem as follows: The input contains the following elements:

1. An unweighted (e, b)-election E with candidate set $C = \{c_1, \ldots, c_m\}$ and a collection V of voters $v_1, \ldots, v_n$, where each voter v_ℓ's preference is represented via an m-dimensional integer vector describing, in a natural way, how many points v_i assigns to each candidate, and where each voter v_ℓ's price is a function $\pi_\ell : C \times C \to \mathbb{N}$.
2. A nonnegative integer B, the budget.

A unit bribery involves asking some voter $v_\ell \in V$ to move a single point that v_ℓ currently assigns to some candidate c_i to another candidate c_j. The cost of such a unit bribery is $\pi_\ell(c_i, c_j)$. Naturally, for each $\ell \in \{1, \ldots, n\}$ and each candidate c_i we have $\pi_\ell(c_i, c_i) = 0$. In the (e, b)-bribery problem we ask if it is possible to perform a set of unit briberies of total cost at most B, such that

1. preferred candidate $p = c_1$ becomes a winner, and
2. the election resulting from our bribery conforms to the requirements of an (e, b)-election, that is, each voter assigns at most b points to each candidate.

We explicitly require that all unit briberies are executed "in parallel." That is, the briber cannot first bribe voter v_ℓ to move a point from some candidate c_i to another candidate c_j and afterward move that same point from c_j to yet another candidate c_q.[4]

[4]The results of this section still hold even if such sequential bribing were legal, but we believe that the "parallel bribery model" is more appropriate.

(However, it would still be legal to move to c_q a point that v_ℓ had assigned to c_j before the bribery. Of course, points are unnamed entities; our requirement of not moving "the same" point twice formally means that briberies are only legal if for each voter v_ℓ they move, within the preference vector of that voter, at most as many points away from each candidate as many that candidate had been assigned by v_ℓ before the bribery.)

The free-form (e, b)-bribery problem is defined analogously, only that voters do not have to assign all their points to candidates (i.e., we are using free-form (e, b)-elections instead of regular (e, b)-elections) and, in addition to other unit briberies, we can bribe voters to either use their unassigned points or to take away points from some candidate without reassigning them to any other one. In order to accomplish this we extend our price functions to incorporate a dummy candidate d representing the slot for unassigned points (naturally, the b-bound does not apply to d).

We define (e, b)-weighted-bribery and free-form (e, b)-weighted-bribery analogously to their unweighted counterparts, only that the voters in the underlying elections are weighted. We use the term "nonuniform bribery" to jointly refer to the type of bribery problems described in this subsection.

4.2.2 Unweighted Nonuniform Bribery Is Easy

The next theorem shows that in almost all natural settings nonuniform bribery in unweighted (e, b)-elections is easy.

Theorem 4.8. *There is an algorithm that solves (e, b)-bribery instances and free-form (e, b)-bribery instances in time polynomial in e* and *the size of the instance.*

Proof. We will first give an algorithm for (e, b)-bribery and then explain how it can be modified to work for free-form (e, b)-bribery.

Our input is an (e, b)-election E, with candidate set $C = \{c_1, \dots, c_m\}$ and voter collection $V = (v_1, \dots, v_n)$, a nonnegative integer B (the budget), and voters' price functions $\pi_1, \dots, \pi_n$. Our goal is to ensure that candidate $p = c_1$ is a winner of the election via a bribery of cost at most B.

Our proof follows via constructing a series of flow networks and computing minimum-cost solutions for them. The intuition here is that the points that the voters assign to candidates are modeled via the units of flow traveling through the network. We design our networks in such a way that minimizing the cost of the flow, in essence, corresponds to finding a minimum-cost bribery that gives candidate $p = c_1$ a prespecified amount of points and ensures that all other candidates have at most as many points.

We know that each candidate receives at most en points. For each nonnegative integer Q between 1 and en our algorithm tests if there is a bribery of cost at most B that ensures that p receives exactly Q points and every other candidate receives at most Q points. Let us now fix a value of Q and show how such a test can be performed.

We form a flow network (K, s, t, c, a), see Section 2.3, with the node set K such that

$$K = \{s, t\} \cup \left(\bigcup_{i=1}^{n} C_i \right) \cup \left(\bigcup_{i=1}^{n} C'_i \right) \cup F,$$

where $F = \{f_1, \dots, f_m\}$ and for each $i \in \{1, \dots, n\}$ we have $C_i = \{c_{i1}, \dots, c_{im}\}$, $C'_i = \{c'_{i1}, \dots, c'_{im}\}$.

We introduce the following capacities and costs for edges in our network. (All unmentioned edges have capacity 0.) For each voter v_ℓ and candidate c_i we have $c(s, c_{\ell i})$ equal to the number of points v_ℓ assigns to c_i before the bribery and $a(s, c_{\ell i}) = 0$.

For each node $c_{\ell i}$ and each node $c'_{\ell j}$ we have $c(c_{\ell i}, c'_{\ell j}) = e$ and $a(c_{\ell i}, c'_{\ell j}) = \pi_\ell(i, j)$. These edges model unit briberies. For each node $c'_{\ell i}$ we set $c(c'_{\ell i}, f_i) = b$ and $a(c'_{\ell, f_i}) = 0$. Finally, we have $c(f_1, t) = Q$, $a(f_1, t) = 0$, and for each $i \in \{2, \dots, t\}$ we have $c(f_i, t) = Q$, $a(f_i, t) = T$, where T is an integer larger than the cost of any possible bribery (e.g., take $T = 1 + en \max_{\ell,i,j} \pi_\ell(c_i, c_j)$).

To perform our test we compute a minimum-cost flow of value en in this network. If such a flow doesn't exist the we disregard this value of Q because there is no way of redistributing voters' points among the candidates so that each candidate receives at most Q points. Let f be a minimum-cost flow that we compute. Since the flow has value en, for each node $c_{\ell i}$ we have $f(s, c_{\ell i})$ equal to the number of points that voter ℓ assigns to candidate c_i in election E. For each two nodes $c_{\ell i}$ and $c'_{\ell j}$, we interpret $f(c_{\ell i}, c'_{\ell j})$ as the number of points that the briber asks v_ℓ to reassign from c_i to c_j. Note that each point traveling on the edge from $c_{\ell i}$ to $c'_{\ell j}$ increases the cost of the flow by $\pi_\ell(c_{\ell i}, c'_{\ell j})$, exactly the price of such a unit bribery. (Recall that if $i = j$ then $\pi_\ell(c_{\ell i}, c'_{\ell j}) = 0$.) Note that for each node $c'_{\ell j}$, $c'_{\ell j}$ receives at most b units of flow because the only edge outgoing from $c'_{\ell j}$, edge $(c'_{\ell j}, f_j)$, has capacity b. For each $j \in \{1, \ldots, m\}$, we interpret the units of flow that enter f_j as the points that candidate c_j receives from all voters after the bribery. Each unit of flow that enters f_j then goes directly to the sink t, at cost T if $j \in \{2, \ldots, m\}$ and at cost 0 if $j = 0$. Thus, the cost of the whole flow is

$$T \cdot (en - p\text{'s score after bribery}) + \text{cost-of-bribery}.$$

T is larger than the cost of any bribery. Thus, each minimum-cost flow f' ensures that $f'(f_1, t)$ is as large as possible. In particular, if there is any flow f'' of value en such that $f''(f_1, t) = c(f_1, t) = Q$ then for each minimum-cost flow f' it also holds that $f(f_1, t) = Q$. So, for our flow f, if $f(f_1, t) \neq Q$ then we can safely disregard this network (in essence, because we have already handled this flow when analyzing smaller values of Q). Via the interpretation of f given above, if $f(f_1, t) = Q$ then the microbribery that f models guarantees that p gets exactly Q points. Since for each candidate c_j it holds that $f(c_j, t) \leq Q$, the bribery modeled by f ensures that p is a winner. In addition, it is easy to see that since f is a minimum-cost flow of value en, the "cost-of-bribery" part of the cost of f is equal to a minimum cost of a bribery that ensures that p gets exactly Q points and that everyone else gets at most Q points.

This way we test, in polynomial-time, for each Q whether there is a nonuniform bribery of cost at most B that ensures that p receives exactly Q points and all other candidates receive at most Q points. In total, the running time of our algorithm is polynomial in the size of our election and e.

It remains to show that free-form (e, b)-bribery also can be solved in a similar manner. The algorithm for free-form (e, b)-bribery works exactly like the one for (e, b)-bribery only that we have to slightly extend our network to account for the fact that each voter may choose to assign only some of the points that he or she has. To do so, we extend the candidate set with a virtual candidate d, the dummy, to whom the voters will assign all the points that they would like to leave unassigned. In the flow network we set the capacities of the edges relating to d so that there are no constraints on the amount of flow that d may receive (but we keep the costs of all these edges intact). It is easy to see that after such modifications of the flow network, our algorithm solves free-form (e, b)-bribery. ❑

The above theorem is quite powerful. In many natural utility-based election systems, such as plurality, veto, or (k-)approval, the number of points to distribute is polynomially bounded in the number of candidates and so, via Theorem 4.8, we get the following corollary.

Corollary 4.9. *Nonuniform bribery can be solved in polynomial time for: plurality, veto, (k-)approval, and utility-based voting where the number of points each voter can distribute is polynomial in the number of candidates (or voters, or both) participating in the election.*

Thus, for most practical purposes, unweighted nonuniform bribery is easy. Naturally, it is interesting to ask if this is also the case for weighted settings. In the next section we show that this is unlikely.

4.2.3 Nonuniform Bribery in Weighted Elections

Let us now focus on *weighted* nonuniform bribery. Unfortunately, when weights come into play, even very basic nonuniform bribery problems become hard. For example, in the next theorem we show that $(1, 1)$-weighted-bribery, that is, nonuniform bribery for weighted plurality elections, is NP-complete even if the values of the price functions are polynomially bounded in the size of the election. This is very interesting because, in our standard bribery model, weighted bribery in plurality with unary-encoded prices is easy (Theorem 4.6).

Theorem 4.10. $(1, 1)$*-weighted-bribery is* NP*-complete even if the values of price functions are polynomially bounded in the size of the election.*

Let us explain the idea of the proof intuitively before we proceed with formal details. In the standard plurality bribery problems (Section 4.1) the briber always asks the voters, or at least those that he or she bribes, to vote for the briber's favorite candidate p. This is a reasonable method of bribing if one wants p to become a winner, but it also has potential real-world downsides: The more people we bribe, the more likely it may be that the malicious attempts will be detected and will work against p. To minimize the chances of that happening we might instead bribe voters to vote not for p but for some other candidate(s). This way p does not get extra votes but might be able to take away enough points from the most popular candidates to become a winner. We will call this setting *negative bribery* because the motivation of the briber is not to get votes for his or her favorite candidate, but to take them away from others.

Definition 4.11. plurality-weighted-negative-bribery *is defined to be the same as* plurality-weighted-bribery*, except that in* plurality-weighted-negative-bribery *the briber is not allowed to bribe voters to vote for the designated candidate p.*

In our proof we will first argue that plurality-weighted-negative-bribery can be modeled as $(1,1)$-weighted-bribery where each price function has values polynomially bounded in the size of the election, and then show that plurality-weighted-negative-bribery is NP-complete.

Proof of Theorem 4.10. plurality-weighted-negative-bribery can easily be expressed as $(1,1)$-weighted-bribery. Given an instance of plurality-weighted-negative-bribery with budget k we form an instance of $(1,1)$-weighted-bribery that is identical (in particular, keeps the same budget) only that the voters' price functions are set such that transferring any point to p costs $k+1$ (and, thus, is unaffordable) and transferring any point to any other candidate costs 1. Thus, to prove the theorem it is enough to show that plurality-weighted-negative-bribery is NP-complete.

Let us now show that plurality-weighted-negative-bribery is NP-complete by giving a reduction from Partition (NP-membership is obvious). Let $s_1, \ldots, s_n$ be a sequence of nonnegative integers. We design an instance of plurality-weighted-negative-bribery such that bribery is possible if and only if $s_1, \ldots, s_n$ can be split into two subsequences that sum up to the same value. Let S be such that $\sum_{i=1}^{n} s_i = 2S$. Our election has three candidates: p, c_1, and c_2, and we have $n+1$ weighted voters:

1. v_0 with weight S, who votes for p, and
2. $v_1, \ldots, v_n$ with weights $s_1, \ldots, s_n$, each of whom votes for c_1.

We want to make p a winner and we allow ourselves to bribe as many candidates as we please. (In particular, we set the bribe limit k to $n+1$.)

Note that the only reasonable bribes are the ones that transfer votes of v_i, $1 \leq i \leq n$, from c_1 to c_2. (Strictly speaking, v_0 could legally be bribed to vote for c_1 or c_2, but that can be safely ignored.) If there is a set $A \subseteq \{1, \ldots, n\}$ such that

$$\sum_{i \in A} s_i = S, \tag{4.1}$$

then we could bribe all voters v_i, $i \in A$, to vote for c_2 and all candidates would be winners. On the other hand, if p can end up a winner by a bribery that does not ask anyone to vote for p, then there is a set A that satisfies Equation (4.1): after a negative bribery p is a winner of our election if and only if each of c_1 and c_2 have vote weight exactly S. However, at the beginning c_1 holds $2S$ vote weight and so a successful bribery needs to transfer exactly S vote weight from c_1 to c_2. This is only possible if (4.1) holds for some A.

To finish the proof, we observe that this reduction can be computed in polynomial time. ❑

Let us now consider weighted nonuniform bribery in approval voting. Approval voting with m candidates is modeled as a free-form $(m,1)$-election. In approval, each voter's 0/1-vector of point assignments is typically called his or her approval vector.

Although nonuniform bribery, as defined in this section, can be applied to approval directly, it seems more natural to consider a restriction where each voter has a separate price for switching each of the entries of his or her approval vector. We will refer to this variant of bribery in approval voting as approval-\$microbribery.

Definition 4.12. *approval-\$microbribery is the problem that takes as input a description of an approval election along with a designated candidate p and a nonnegative integer k, and asks whether it is possible to make p a winner by at most k entry changes (total) in the approval vectors. Changing each entry of an approval vector may have a different price.*

Naturally, we can consider a weighted version, approval-weighted-\$microbribery, by allowing the voters to be weighted.

It is easy to see that microbribery for approval is a special case of nonuniform bribery for approval, where the only affordable prices are those of moving a point from a real candidate to the dummy one and the other way round. Thus, via Theorem 4.8, approval-\$microbribery is in P. On the other hand, we show that approval-weighted-\$microbribery is NP-complete, which implies that free-form $(m, 1)$-weighted-bribery, where m is the number of candidates, is NP-complete.

Theorem 4.13. *approval-weighted-\$microbribery is* NP*-complete.*

Proof. It is immediate that approval-weighted-\$microbribery is in NP. To show NP-hardness, we will construct a reduction from Partition. Let $s_1, \ldots, s_n$ be a sequence of nonnegative integers and let $\sum_{i=1}^{n} s_i = 2S$. We construct an election E with candidates p and c and $n + 1$ voters, $v_0, \ldots, v_n$, with the following properties.

1. v_0 has weight S, approves only of p, and changing any of v_0's approvals costs $S+1$.
2. v_i, for $1 \leq i \leq n$, has weight s_i, approves only of c, changing v_i's approval for p costs s_i, and changing v_i's approval for c costs $S + 1$.

We claim that p can be made a winner by a bribery of cost at most S if and only if there is a set $A \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in A} s_i = S$.

First suppose that p can be made a winner by a bribery of cost at most S. Briberies of cost up to S are exactly those that involve bribing some subcollection of $v_1, \ldots, v_n$

to approve of p. Since before bribery p has S approvals and c has $2S$ approvals, our bribery needs to give p at least S extra approvals. Since changing each v_i's approval of p costs s_i, and the weight of each v_i is also s_i, it follows that via a successful bribery p gains exactly S approvals, and that the weights of the bribed voters in $v_1, \ldots, v_n$ add up to exactly S. This implies that the sequence $s_1, \ldots, s_n$ can be partitioned into two subsequences that each sum to S.

On the other hand, assume there is a set $A \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in A} s_i = S$. Then we can bribe voters v_i, $i \in A$, to approve of p. As a result, both p and c will have vote weight $2S$ and both of them will be winners. Our reduction can be computed in polynomial time and thus the theorem is proved. ❑

The hardness of approval-weighted-\$microbribery strictly relies on the fact that both prices and weights can be large (since they are represented in binary). If either prices or weights are small (in particular, if they are represented in unary) then the problem becomes easy. We let approval-weighted$_{\text{unary}}$-\$microbribery be a variant of approval-weighted-\$microbribery with weights represented in unary, and we let approval-weighted-\$microbribery$_{\text{unary}}$ be a variant of approval-weighted-\$microbribery with prices represented in unary.

Theorem 4.14. *Both approval-weighted$_{\mathit{unary}}$-\$microbribery and approval-weighted-\$microbribery$_{\mathit{unary}}$ are in* P.

Proof. The polynomial-time algorithm we provide is based on the observation that in both approval-weighted$_{\text{unary}}$-\$microbribery and approval-weighted-\$microbribery$_{\text{unary}}$ getting vote weight for the favorite candidate can be (carefully) treated separately from demoting the other candidates.

We can divide any bribery into two phases: First, we bribe voters to approve of p, our favorite candidate, and second, we bribe enough voters to decline their approvals of candidates that still defeat p. There are only polynomially many relevant vote weights that p may obtain by bribery, so we can try them all.

Consider a bribery instance with election $E = (C, V)$, preferred candidate p, and budget k. For a candidate c, a price b, and a subset of voters V', we define $\mathit{heaviest}(V', c, b)$ to be the highest vote weight of voters in V' whose approval of c can be switched by spending at most b dollars. Similarly, for a candidate c, vote weight w, and a subset of voters V', we define $\mathit{cheapest}(V', c, w)$ to be the lowest price that can switch the approval-of-c of voters in V' that have total weight at least w. In our proof we only use sets V' where either all voters approve of c or all voters disapprove of c. Note that $\mathit{heaviest}(V', c, b)$ is defined for all $b \geq 0$ and that $\mathit{cheapest}(V', c, w)$ is defined for all $w \leq \omega(V')$. As in Section 4.1, *heaviest* can easily be computed in polynomial time in the unary prices case and *cheapest* can easily be computed in polynomial time in the unary weights case. In addition, *cheapest* can be computed in polynomial time in the unary prices case. Note that

$$\mathit{cheapest}(V', c, w) = \min\{b \mid \mathit{heaviest}(V', c, b) \geq w\}.$$

Since there are only polynomially many prices to try, this can be done in polynomial time.

Figure 4.3 gives pseudocode for the procedure *UnaryPricesApproval*, which decides approval-weighted-\$microbribery$_{\text{unary}}$. The procedure simply tries all relevant weights that p could obtain by bribery and tests whether it is possible, for any of them, to bring the other candidates down to vote weight at most that of p without exceeding the budget. The procedure is correct because of the separation we achieved between the issue of bribing voters to approve of p and the issue of bribing them not to approve of some other candidate. Also, as *cheapest* and *heaviest* are computable in polynomial time, the procedure works in polynomial time. An analogous procedure decides the unary weights case: Simply change the line "**for** $b = 0$ **to** k **do**" to "**for** $w = 0$ **to** $\omega(V')$ **do**" and the line "$w = \mathit{heaviest}(V', p, b)$" to "$b = \mathit{cheapest}(V', p, w)$." ❑

To make things more interesting: In our standard model of bribery, approval is hard even in the simplest setting (without prices and without weights).[5]

Theorem 4.15. approval-bribery *is* NP*-complete.*

[5]Which of the above-discussed bribery models for approval is most appropriate depends on the

```
procedure $UnaryPricesApproval(E = (C, V, p, k))$
begin
    if $k \geq \pi(V)$ then accept;
    $V' = \{v \mid v \in V$ and $v$ does not approve of $p\}$;
    for $b = 0$ to $k$ do
    begin
        $w = heaviest(V', p, b)$;
        $r = score_E(p) + w$;
        $k' = k - b$;
        for $c \in C - \{p\}$ do
        begin
            $V'_c = \{v \mid v \in V$ and $v$ approves of $c\}$;
            if $score_E(c) > r$ then
                $k' = k' - cheapest(V'_c, c, score_E(c) - r)$;
        end
        if $k' \geq 0$ then accept;
    end
    reject;
end
```

Figure 4.3: The procedure *UnaryPricesApproval*.

Proof. Clearly, approval-bribery is in NP. NP-completeness follows from a reduction from X3C.

Let $B = \{b_1, \ldots, b_{3t}\}$ and let $\mathcal{S} = \{S_1, \ldots, S_m\}$ be a family of three-element subsets of B. Without loss of generality, we assume that $m \geq t$; otherwise an exact cover is impossible. For each i, $1 \leq i \leq 3t$, let ℓ_i be the number of sets S_j that contain b_i. On

setting. For example, microbribery seems more natural when we look at the web and treat web pages as voting by linking to other pages. It certainly is easier to ask a webmaster to add/remove a link than to completely redesign the page.

input $(B, \mathcal{S})$ we form an election $E = (C, V)$, where $C = \{p\} \cup B$, and voter collection V contains the following voters.

1. For each $S_i \in S$ there is a voter v_i who approves exactly of the members of S_i.

2. For each b_i we have $m - \ell_i + 1$ voters who approve only of b_i.

3. We have $m - t$ voters who approve only of p.

Note that p gets $m - t$ approvals and that each b_i, $1 \leq i \leq 3t$, gets $m + 1$ approvals. We claim that p can be made a winner by bribing at most t voters if and only if B has an exact cover by sets in $\mathcal{S}$.

First assume that there is a set A such that $\|A\| = t$ and $\bigcup_{i \in A} S_i = B$. To make p a winner, bribe each v_i such that $i \in A$ to approve only of p. As a result p gets t additional approvals and each b_i loses exactly one approval. Thus, all candidates are winners. On the other hand, assume there is a bribery of at most t voters that makes p a winner. Each bribed voter contributes at most one additional approval for p. Thus, p will get at most m approvals. Each candidate in B has $m + 1$ approvals, and our bribery needs to take away at least one approval from each candidate in B. Since we bribe at most t voters, this can only happen if we bribe t voters v_i that correspond to a cover of B. This reduction can be computed in polynomial time and the proof is complete. ❑

By the above discussion, we cannot hope for a general efficient algorithm handling (e, b)-weighted-bribery for e, b polynomially bounded in the size of the input election. We need to seek either a general NP-hardness proof that would cover all reasonable values of e and b or, alternatively, special cases of weighted nonuniform bribery for which polynomial-time algorithms exist.

4.3 Conclusions and Research Directions

In this chapter we have presented a comprehensive study of the complexity of bribery in plurality elections and nonuniform bribery in utility-based systems. In particular, in Section 4.1 we have shown that bribery in plurality elections is almost always easy, except for the setting in which voters have both weights and (binary encoded, relatively large) prices. This last result follows via a proof that involves elections with two candidates only. Since many natural election systems reduce to plurality when only two candidates are involved, our proof is in fact quite general.

We have also shown that even though finding *optimal* briberies for the case of weighted-and-priced plurality elections is hard, finding approximate solutions is easy, and that in fact there is an FPTAS for finding minimum-cost briberies in this setting.

In Section 4.2 we have considered nonuniform bribery in utility-based elections. We have shown that in the unweighted case bribery is easy for utility-based weighted election systems where the number of points that the voters have to distribute is polynomially bounded. However, for the case of weighted elections we have shown examples of nonuniform bribery problems that are NP-complete. An interesting open research direction is to provide a general classification result for the complexity for nonuniform bribery in weighted elections (especially in the case where the values of price functions are encoded in unary).

5 Manipulating Scoring Protocols

Scoring protocols constitute one of the most important and natural classes of election rules. Some of the oldest practically used election systems, such as plurality and Borda count, as well as many other natural systems, e.g., k-approval and veto, can be viewed as families of scoring protocols. To mention just two examples of practical modern-day applications of scoring protocols, Borda count is used in certain political elections (for example, in Slovenia) and a scoring rule with vector $(12, 10, 8, 7, 6, \dots, 1, 0, \dots, 0)$ is used for voting in the Eurovision Song Contest. Simplicity, naturalness, and popularity of election systems based on scoring protocols make it important to study their computational properties.

The starting point for the discussion in this chapter is the following theorem of Hemaspaandra and Hemaspaandra (see also (PR07) and (CSL07)) that fully classifies the hardness of weighted manipulation in scoring protocols.

Theorem 5.1 (Hemaspaandra and Hemaspaandra (HH07))**.** *Let* $\alpha = (\alpha_1, \dots, \alpha_m)$ *be a scoring protocol. If it is not the case that* $\alpha_2 = \alpha_3 = \cdots = \alpha_m$*, then* α-weighted-manipulation *is* NP*-complete; otherwise, it is in* P*.*

We extend this theorem in the following two directions. First, in Section 5.1, we obtain similar classification results for the case of bribery in scoring protocols for each combination of priced-vs-unpriced with weighted-vs-unweighted voters. From our bribery results we derive algorithms for manipulation in scoring protocols for the cases where voters are either unweighted or have weights encoded in unary (cases not covered by

Hemaspaandra and Hemaspaandra (HH07)). Then in Section 5.2, we refine Theorem 5.1 by showing that for weighted manipulation problems in many scoring protocols there are fully polynomial-time approximation schemes. In so doing, we give a fairly general framework for approximate analysis of manipulation, bribery, and control problems.

5.1 The Complexity of Bribery in Scoring Protocols

Our goal in this section is to obtain a full classification of scoring protocols with respect to their hardness of bribery. As we have already seen in Section 4, the complexity of bribery may strongly depend on whether the voters are weighted and whether they have price tags. Thus, in this section, we are naturally interested in seeing how the complexity of bribery in scoring protocols depends on the type of voters involved.

Instead of obtaining our bribery results from scratch, we would ideally like to derive them from Theorem 5.1. This is particularly easy for the case of weighted-and-priced voters, as in this case we can use Theorem 3.4 which says that every manipulation problem reduces to an analogous priced bribery problem. Thus, combining Theorem 3.4, Theorem 5.1, and our results on the complexity of bribery for plurality from Section 4.1, we obtain the following complete classification of scoring protocols with respect to the complexity of weighted-\$bribery.

Theorem 5.2. *For each scoring protocol* $\alpha = (\alpha_1, \ldots, \alpha_m)$*, if* $\alpha_1 = \alpha_m$ *then* α-weighted-\$bribery *is in* P*; otherwise it is* NP*-complete.*

Proof. We consider three cases.

1. $\alpha_1 = \cdots = \alpha_m$.
2. $\alpha_1 > \alpha_2 = \cdots = \alpha_m$.
3. All other settings.

In the first case, $\alpha_1 = \cdots = \alpha_m$, α-weighted-\$bribery is trivially in P as all candidates are always tied. For the remaining two cases, note that α-weighted-\$bribery is clearly in NP. It remains to show NP-hardness.

In the second case, $\alpha_1 > \alpha_2 = \cdots = \alpha_m$, we can employ the proof of Theorem 4.2. Theorem 4.2 shows NP-hardness for $(1, 0)$-weighted-\$bribery. It is easy to see that for all $m \geq 2$ we can pad this reduction with $m - 2$ candidates that are never ranked first to obtain NP-hardness for $(1, \overbrace{0, \ldots, 0}^{m-1})$-weighted-\$bribery. Note that our α describes elections equivalent to plurality (i.e., a candidate is a winner of an α election if and only if he or she would also be a winner of the $(1, \overbrace{0, \ldots, 0}^{m-1})$ election with the same voters and candidates; see (HH07, Observation 2.2)). Thus, we get NP-completeness of α-weighted-\$bribery for this case since we do have at least two candidates.

The third case follows by combining Theorem 3.4 with Theorem 5.1. Since α-weighted-manipulation many-one reduces to α-weighted-\$bribery and α-weighted-manipulation is NP-complete in this case, we have that α-weighted-\$bribery is NP-hard. This exhausts all cases. ❑

Theorem 5.2 applies to weighted-\$bribery, but of course it is also interesting to ask what happens in the case where voters do not have prices. Does bribery remain NP-complete? Can we express the constraints of manipulation without using such direct embedding as above? The following theorem shows that the answer is "Yes, but in fewer cases."

Theorem 5.3. *For each scoring protocol $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_m)$, if $\alpha_2 = \alpha_3 = \cdots = \alpha_m$ then* α-weighted-bribery *is in* P*; otherwise it is* NP*-complete.*

If $\alpha_2 = \alpha_3 = \cdots \alpha_m$ then either α-weighted-bribery is trivially in P (if $\alpha_1 = \cdots = \alpha_m$) or can be solved using the algorithm for plurality-weighted-bribery. The core of the proof is to show NP-hardness. It would be nice to do so by reducing from the corresponding manipulation problems (which share the characterization's boundary line regarding the "α"s). This seems not to work, but in Lemma 5.5 we construct such a reduction that has the right properties whenever its inputs satisfy an additional condition, namely, that the weight of the lightest manipulating voter is at least double that of the heaviest nonmanipulator. This would suffice if the thus-restricted manipulation problem were NP-hard. Lemma 5.6 shows that the thus-restricted manipulation problem *is*

NP-hard. It does so by examining the manipulation-dichotomy proof of Hemaspaandra and Hemaspaandra (HH07) and noting that if we apply their reduction to Partition$'$ (see Section 2.2) rather than to Partition then we can guarantee the restriction mentioned above.

Definition 5.4. *By* α-weighted-manipulation$'$ *we mean the manipulation problem* α-weighted-manipulation *with the restriction that each manipulative voter has weight at least twice as high as the weight of the heaviest of the nonmanipulative voters. Each instance where the restriction is violated is considered not to be an element of* α-weighted-manipulation$'$*.*

Lemma 5.5. *Let* $\alpha = (\alpha_1, \ldots, \alpha_m)$ *be a scoring protocol.* α-weighted-manipulation$' \leq_{\mathrm{m}}^{\mathrm{p}}$ α-weighted-bribery*.*

Proof. Without loss of generality we can assume that $\alpha_m = 0$. If $\alpha_m \neq 0$ then we can consider scoring protocol $\alpha' = (\alpha_1 - \alpha_m, \alpha_2 - \alpha_m, \ldots, \alpha_m - \alpha_m)$ instead. We give a reduction from α-weighted-manipulation$'$ to α-weighted-bribery. Let our input instance M contain election $E = (C, V)$ and a collection S of manipulators, who want to ensure that their preferred candidate $p \in C$ is a winner. We form an instance B of bribery with election $E' = (C, V')$, preferred candidate p, and budget $\|S\|$. We set $V' = V \cup S'$, where S' is the collection of voters S with arbitrary preference lists that rank p last. We assume that M fulfills α-weighted-manipulation$'$'s requirements regarding relative weights of voters in V and S. If not, we output some fixed B that has no successful briberies.

Clearly, if a manipulation is possible within M then some bribery works for B. We show that the other direction also holds by arguing that if a successful bribery within B exists, then there is a successful bribery that affects only voters in S'. This implies that one can assign preference lists to voters in S so that p is a winner of $(C, V \cup S)$.

Let us assume that there is some way of bribing at most $\|S\|$ voters in V' so that p becomes a winner. If all the bribed voters are in S' then the theorem is proven. Otherwise, select some bribed voter $v \in V' - S'$. By bribing v, p gains at most $(\alpha_1 + \alpha_1) \cdot \omega(v)$ points over each candidate $c \neq p$. (The first α_1 is because p can get at most α_1 additional points by this bribery, and the second α_1 is because c can lose at most α_1 votes.) However, if instead of bribing v we would bribe some voter v' in S', p would gain at least $\alpha_1\omega(v')$ points over each c. (We would bribe v' to put p as his or her most preferred candidate and shift all other candidates back.) Since it holds that $\omega(v') \geq 2\omega(v)$, we might just as well bribe v' instead of v, and p would still be a winner. Thus, if p can be made a winner, then p can be made a winner by bribing only voters in S'. This reduction can easily be computed in polynomial time and the proof is complete. ❑

It remains to show that the restricted version of manipulation is NP-complete for each of the scoring protocols for which the nonrestricted version is.

Lemma 5.6. *If $\alpha = (\alpha_1, \ldots, \alpha_m)$ is a scoring protocol such that it is not the case that $\alpha_2 = \alpha_3 = \cdots = \alpha_m$, then* α-weighted-manipulation$'$ *is* NP*-complete.*

Proof. Let $\alpha = (\alpha_1, \ldots, \alpha_m)$ be a scoring protocol such that $\alpha_2 \neq \alpha_m$. We will use Hemaspaandra and Hemaspaandra's proof of Theorem 5.1 (HH07) to show the NP-completeness of α-weighted-manipulation$'$. Clearly, α-weighted-manipulation$'$ is in NP so we only need to prove its NP-hardness.

Hemaspaandra and Hemaspaandra's proof of Theorem 5.1 (HH07) reduces Partition (restricted to positive integers) to α-weighted-manipulation. A close inspection of that proof[1] shows that there exist constants c and $d \geq 2$ that depend only on α such that for every sequence of positive integers $s_1, \ldots, s_n$ such that $\sum_{i=1}^{n} s_i = 2S$, the Hemaspaandra-Hemaspaandra reduction outputs a manipulation problem that has the following properties.

1. Each nonmanipulative voter has weight at most cS, and

2. the weights of the manipulative voters are $ds_1, ds_2, \ldots, ds_n$.

We will use these facts to provide a reduction from Partition$'$ to α-weighted-manipulation$'$.

[1]We do not repeat that proof here. Interested readers are referred to (HH07).

Our reduction works as follows. Let $s_1, \ldots, s_n$ be the input sequence of nonnegative integers, $\sum_{i=1}^{n} s_i = 2S$, such that for each i, $1 \leq i \leq n$, it holds that $s_i \geq \frac{2}{2+n}S$. (If these conditions do not hold then, as is our convention, we return a fixed string not in α-weighted-manipulation$'$.) Without loss of generality, we assume that $S > 0$, and thus $s_1, \ldots, s_n$ are positive integers. Let f be the reduction given by the proof of Theorem 5.1 from (HH07). We compute $f((s_1, \ldots, s_n)) = M$, where M is a α-weighted-manipulation instance with an election $E = (C, V)$, a collection T of manipulators, and a preferred candidate $p \in C$. Reduction f works for general Partition and so, since we already checked the special properties required by Partition$'$, it has to work correctly for our input. That is, $s_1, \ldots, s_n$ can be partitioned if and only if there is a way for the manipulators in T to ensure p's victory. Unfortunately, we cannot just output M as it does not necessarily fulfill the condition on voters' weights. Recall that we have to ensure that each manipulative voter has weight at least twice as high as the weight of the heaviest of the nonmanipulative voters. Let $s_{\min} = \min\{s_j \mid 1 \leq j \leq n\}$. In M, the least weight of a voter in T is exactly $ds_{\min}$, and the highest weight of a voter in V is at most cS. However, we can split each voter v in V. The weights of the voters who do not participate in the manipulation are irrelevant as long as the total weight of voters with each given preference order does not change. Thus, we can replace a voter with high weight by several other voters with the same preference order but with lower weights. In our case, we need to make sure that each nonmanipulative voter has at most weight $\frac{1}{2}ds_{\min}$. Since the heaviest of the nonmanipulative voters has weight at most cS, we need to replace each voter $v \in V$ with at most

$$\left\lceil \frac{cS}{\lfloor \frac{1}{2}ds_{\min} \rfloor} \right\rceil \tag{5.1}$$

voters, each of weight at most $\frac{1}{2}ds_{\min}$. Since $d \geq 2$, $S > 0$, $s_{\min}$ is a positive integer, and $\frac{2}{2+n}S \leq s_{\min}$, we can bound (5.1) from above by

$$\left\lceil \frac{cS}{\lfloor \frac{1}{2}ds_{\min} \rfloor} \right\rceil \leq \left\lceil \frac{cS}{s_{\min}} \right\rceil \leq \left\lceil \frac{cS}{\frac{2S}{2+n}} \right\rceil = \left\lceil \frac{c(n+2)}{2} \right\rceil,$$

which is clearly polynomially bounded in n. Thus, the splitting of voters can easily be performed in polynomial time, and since it does not change the result of manipulation, the theorem is proven. ❑

The proof of Theorem 5.3 simply combines Lemmas 2.3, 5.5, and 5.6.

Theorem 5.2 and Theorem 5.3 show that weighted-bribery and weighted-\$bribery within scoring protocols are, in most cases, difficult. It is interesting to ask whether having voters who have prices but are not weighted also yields such hardness results. As Theorem 5.7 shows, this is not the case.

Theorem 5.7. *Let* $\alpha = (\alpha_1, \ldots, \alpha_m)$ *be a scoring protocol.* α-\$bribery *is in* P.

Proof. We will give a polynomial-time algorithm for α-\$bribery. Our input is election $E = (C, V)$, preferred candidate $p \in C$, and budget k. First, observe that by considering scoring protocol $\alpha = (\alpha_1, \ldots, \alpha_m)$ we, by definition, limit ourselves to a scenario with m candidates, where m is a fixed constant. This implies that there are only a constant number of different preference orders, $o_1, \ldots, o_{m!}$, that the voters might have. We partition V into sets $V_1, V_2, \ldots, V_{m!}$ such that each V_i contains exactly the voters with preference order o_i. Some V_i's might be empty and each V_i has at most n elements, where $n = \|V\|$.

A bribery within E can be described by giving two sequences of integers, $b_1, \ldots, b_{m!}$ and $d_1, \ldots, d_{m!}$, such that $0 \leq b_i \leq \|V_i\|$ and $0 \leq d_i \leq n$, for $1 \leq i \leq m!$, and

$$\sum_{i=1}^{m!} b_i = \sum_{i=1}^{m!} d_i.$$

Each b_i says how many voters from V_i we are bribing. It is sufficient to just give the numbers b_i since we want to bribe only the cheapest members of each V_i. After we bribe these $b = \sum_{i=1}^{m!} b_i$ voters, we need to decide what preferences to assign to them. This is described by the sequence $d_1, \ldots, d_{m!}$: Each d_i says how many of the b voters will be assigned to have preferences o_i. Since the voters are indistinguishable, specifying these numbers is enough.

It remains to observe that there are at most $n^{m!}$ sequences $b_1, \ldots, b_{m!}$ and there are at most $n^{m!}$ sequences $d_1, \ldots, d_{m!}$ for each b. Thus, there are at most $n^{2(m!)}$ sequences to try out. For each pair of sequences it is easy to check whether after performing the described bribery p becomes a winner and whether the budget is not exceeded. Thus, α-\$bribery is in P. ❑

There are a few issues raised by the above proof. The first one is that the proof works for essentially all elections with a fixed number of candidates (as long as the outcome of elections does not depend on the order of votes, but only on their values). It is natural to ask why prices and weights exhibit such differing behavior, that is, why α-weighted-bribery is NP-complete for almost all scoring protocols α, and α-\$bribery is in P for each scoring protocol α. One answer is that in the weighted case the voters retain their individuality—their weights—throughout the whole process of bribery. On the other hand, in the priced case the voters are disassociated from their prices as soon as we decide to bribe them. If we decide to bribe a particular priced voter then we simply need to add his or her price to our total budget, but from then on the voter is indistinguishable from all the other bribed ones. It is precisely this observation that facilitated the proof of Theorem 5.7.

The second issue is the disappointing running time of the given algorithm. While $n^{O(m!)}$ is a polynomial in our setting, one would certainly prefer to have an algorithm whose time complexity did not depend on m in this way. In particular, it would be nice to have an algorithm with running time polynomial in $n + m$. However, if such an algorithm existed then P = NP. This follows from the proof of the fact that approval-bribery is NP-complete (Theorem 4.15). In that proof we showed how to reduce X3C to approval-bribery in such a way that each voter approves of at most 3 candidates. If there were a polynomial p and an algorithm that ran in time $p(\|C\|+\|V\|)$ for every scoring protocol α, then we could solve X3C by reducing it to approval-bribery and then embedding that approval-bribery problem in an α-bribery problem for some $\alpha = (1, 1, 1, 0, \ldots, 0)$, possibly adding some dummy candidates. This embedding is straightforward so we do not describe it in detail.

Let $\alpha = (\alpha_1, \ldots, \alpha_m)$ be a scoring protocol such that it is not the case that $\alpha_2 = \cdots = \alpha_m$. By Theorem 5.3 we know that α-weighted-bribery is NP-complete. We also know, by Theorem 5.7, that α-\$bribery is in P. It clearly holds that α-weighted-\$bribery is NP-complete, but it is interesting to ask whether the NP-completeness of α-weighted-bribery and α-weighted-\$bribery holds because of the possibly exponentially-large values of the weights, or do these problems remain NP-complete even if the weights are encoded in unary? It turns out, by the following theorem, that high weight values are necessary for NP-completeness.

Theorem 5.8. *Let* $\alpha = (\alpha_1, \ldots, \alpha_m)$ *be a scoring protocol.* α-weighted$_{\text{unary}}$-\$bribery *is in* P*.*

Proof. Let $\alpha = (\alpha_1, \ldots, \alpha_m)$ be a scoring protocol. The proof of this theorem cashes in on the same observation as that made in the proof of Theorem 5.7: There are only finitely many different preference orders, and there are only polynomially many substantially different ways of bribing.

Consider an instance of bribery with an election $E = (C, V)$, preferred candidate p, and budget k. By choice of the scoring protocol α, $\|C\| = m$. Let $o_1, \ldots, o_{m!}$ be all the different possible preference orders over C. We partition V into $m!$ disjoint sets $V_1, \ldots, V_{m!}$ such that each V_i contains exactly the voters with preference order o_i. A bribery within E can be described by a sequence of $m!$ vectors $b_i = (b_{i,1}, b_{i,2}, \ldots, b_{i,m!})$, $1 \leq i \leq m!$, such that for each i, j, $1 \leq i, j \leq m!$, $b_{i,j}$ is a nonnegative integer and for each i, $1 \leq i \leq m!$, we have

$$\sum_{j=1}^{m!} b_{i,j} = \omega(V_i).$$

The interpretation of a vector b_i is that voters in V_i can be partitioned into $m!$ collections $V_{i,1}, \ldots, V_{i,m!}$ such that $\omega(V_{i,j}) = b_{i,j}$, with the intention of bribing voters in $V_{i,j}$ to change their preference lists to o_j. When $i \neq j$ this bribery has some price, and when $i = j$ it is for free as nothing really needs to be done. Note that not all vectors are realizable; not every splitting of vote weight $\omega(V_i)$ can be achieved. The rest of this proof is devoted to developing a method for evaluating whether a given split is possible and computing what its minimal cost is. There are only $(\omega(V)^{m!})^{m!}$ ways of selecting vectors $b_1, \ldots, b_{m!}$ so if we can test whether a given vector is realizable (and compute the minimal price for its realization), then we can simply try all sequences of vectors and test whether any of them both makes p a winner and has its total cost fall within the budget.

Let $w = (w_1, \ldots, w_{m!})$ be a sequence of nonnegative integers. By $V_i'(w_1, \ldots, w_{m!})$ we mean the following set of $m!$-element sequences of subcollections of V_i:

$$V_i'(w) = \{(V_{i,1}, \ldots, V_{i,m!}) \mid (V_i = \bigcup_{j=1}^{m!} V_{i,j}) \wedge (\forall 1 \leq j \leq m!)[\omega(V_{i,j}) = w_j]\}.$$

For each vector w we define

$$g_i(w) = \begin{cases} \min\{\rho \mid (\exists (V_{i,1}, \ldots, V_{i,m!}) \in V_i'(w))[\rho = \sum_{j \neq i} \pi(V_{i,j})]\} & \text{if } V_i'(w) \neq \emptyset, \\ \infty & \text{otherwise.} \end{cases}$$

That is, $g_i(w)$ gives the lowest price for bribing the voters in V_i according to weight vector $(w_1, \ldots, w_{m!})$. We can compute $g_i(w)$ in polynomial time using dynamic programming techniques. Let us rename the candidates so that $V_i = \{v_1, \ldots, v_t\}$ and let $g_{i,\ell}(w)$ be the same as $g_i(w)$ except restricted to voters $v_\ell, \ldots, v_t$. Thus, $g_{i,1}$ is exactly g_i. Naturally, the following boundary condition holds for $g_{i,t+1}$.

$$g_{i,t+1}(w_1, \ldots, w_{m!}) = \begin{cases} 0 & \text{if } w_1 = w_2 = \cdots = w_{m!} = 0, \\ \infty & \text{otherwise.} \end{cases}$$

We can compute values of $g_{i,\ell}(w_1, \ldots, w_{m!})$ using dynamic programming and the observation that $g_{i,\ell}(w_1, \ldots, w_{m!})$ is equal to the minimum of the following:

$$g_{i,\ell+1}(w_1 - \omega(v_\ell), w_2, \ldots, w_{m!}) + \pi(v_\ell),$$

$$g_{i,\ell+1}(w_1, w_2 - \omega(v_\ell), w_3, \ldots, w_{m!}) + \pi(v_\ell),$$

$$\ldots$$

$$g_{i,\ell+1}(w_1, \ldots, w_{m!-1}, w_{m!} - \omega(v_\ell)) + \pi(v_\ell), \text{ and}$$

$$g_{i,\ell+1}(w_1, \ldots, w_{i-1}, w_i - \omega(v_\ell), w_{i+1}, \ldots, w_{m!}).$$

Note that the last of the values handles the fact that if we bribe v_ℓ to report preference order o_i then we actually do not need to pay him or her; v_ℓ already has preference order o_i. Otherwise, we need to decide which of the $m! - 1$ other preference orders we ask v_ℓ to report, and we need to pay for this change. Clearly, using this rule and the above boundary condition we can compute $g_{i,1}(w)$, and thus $g_i(w)$, in time polynomial in $\omega(V)$. Since $\omega(V)$ is polynomial in the size of the input, this completes the proof. ❑

Now, via Theorem 3.4 and the above result, we immediately obtain the complexity of manipulation in scoring protocols where the voters are either unweighted or have unary-encoded weights.

Corollary 5.9. *For any scoring protocol α,* α-weighted$_{\text{unary}}$-manipulation *is in* P.

Proof. By Theorem 3.4, scoring protocol α, α-weighted$_{\text{unary}}$-manipulation many-one reduces to α-weighted$_{\text{unary}}$-\$bribery, and as the latter is in P, so is the former. ❑

Certain scoring protocols have natural generalizations to an arbitrary number of candidates (e.g, Borda, veto). Although our results above do not formally imply simplicity of bribery for such election systems (as we need a single P algorithm to work in all cases), such results can often be easily obtained "by hand." For example, Theorem 5.3 implies that veto-weighted-bribery is NP-complete even for 3 candidates. Yet the following result shows that the difficulty of bribery for veto voting comes purely from the weighted votes.

Theorem 5.10. veto-bribery *is in* P.

Proof. The proof of this theorem is essentially the same as that of Theorem 4.1. We can view veto elections as elections in which every voter vetoes one candidate, and each candidate with the least number of vetoes wins.

Thus, given a bribery instance with election $E = (C, V)$, preferred candidate $p \in C$, and budget k, we keep on bribing voters that veto p and ask them to veto a candidate that, at that time, has the least number of vetoes. If after at most k bribes p is a winner then we accept; otherwise we reject. A simple inductive argument shows this is a correct strategy. The algorithm clearly runs in polynomial time. ❑

Zuckerman, Procaccia, and Rosenschein (ZPR08), using a very different approach, showed that veto-manipulation is in P.

5.2 Approximately Manipulating Scoring Protocols

In this section we refine Theorem 5.1 by studying the existence of fully polynomial-time approximation schemes (FPTASes) for weighted manipulation in scoring protocols. In particular, for each scoring protocol $(\alpha_1, \ldots, \alpha_m)$ where $\alpha_1 > \alpha_2$ we give an FPTAS for the optimization variant of α-weighted-manipulation problem. The existence of these FPTASes shows that, in spite of Theorem 5.1, weighted manipulation in (our subclass of) scoring protocols may be computationally easy in practice, provided that we are willing to accept approximate solutions.

5.2.1 Definitions and Discussion

To discuss approximate solutions for manipulation (or bribery, or any other decision problem), we need to establish an appropriate goal function. In the case of bribery, a very natural approach is to measure the cost of making our favorite candidate a winner. We took this approach in Theorem 4.7.

In case of manipulation, coming up with a natural goal function is not as straightforward. For example, we can think of unweighted manipulation problems as follows. We are given an election $E = (C, V)$ and a preferred candidate $p \in C$, and we are asked to find a smallest possible set of additional voters S (each with preferences over C) such that p is a winner of election $(C, V \cup S)$. That is, we are interested in finding a smallest set of manipulators such that these manipulators can ensure p's victory by casting appropriate votes. This approach was suggested and used by Zuckerman, Procaccia, and Rosenschein (ZPR08) in their study of manipulation within Borda and several other voting rules. However, this approach is not completely satisfactory. For example, it is not clear how to translate this approach to the world of weighted manipulation (though, like (ZPR08), we use a workaround for this issue later in this section). More importantly, the "smallest-manipulator-set" approach does not tell us how to most effectively use the manipulators that we do have, but rather says how many manipulators we need to gather. While these two questions are very related, one can easily see how the ability to solve the former implies the ability to solve the latter (at least if we limit ourselves to having at most polynomially many manipulators), but the other direction is not apparent. Finally, the approach of (ZPR08) cannot be applied to settings where the preferred candidate is already a winner. In such settings it might seem unreasonable to even consider manipulation, but since our information regarding other voters might be imperfect, the manipulators still might want to cast votes that benefit their preferred candidate most. Thus, in the remainder of this section, we introduce and study a measure of success of manipulation that, in essence, quantifies how much our preferred candidate benefits from the manipulative votes. The applicability of our approach extends beyond scoring protocols and beyond manipulation, but here we focus on manipulation in scoring protocols.

Let $\mathcal{E}$ be an election system (e.g., a scoring protocol). Each input to the (possibly weighted) manipulation problem for $\mathcal{E}$ contains an election $E = (C, V)$, a preferred candidate $p \in C$, and a collection of manipulative voters S. Voters in S do not yet have any preference lists assigned. A solution sol to such an instance of manipulation is a collection S' of voters that is identical to S, only that each voter in S' is assigned some preference list. By $sol(E)$ we mean $(C, V \cup S')$. Clearly, each such instance of manipulation has at least one solution.

Our goal function works as follows. Let $E = (C, V)$ be our input election, $p \in C$ be our preferred candidate, and sol be a solution to the input manipulation problem. We define p's *performance* in election E as

$$perf_E(c) = score_E(c) - \max_{d \in C}(score_E(d)).$$

$perf_E(p)$ tells us either how far p is from winning, or by how much he or she is winning, depending on whether $perf_E(p)$ is negative or positive. Our goal is to make $perf_E(p)$ as large as possible. By $\beta(E, sol)$ we mean the increase in $perf_E(p)$ that we obtain via applying solution sol. That is,

$$\beta(E, sol) = perf_{sol(E)}(p) - perf_E(p).$$

We propose to set the goal in the optimization variant of manipulation to be to maximize $\beta(E, sol)$. At first, this goal may seem quite unnatural, so let us briefly explain why it is, in fact, very useful. First, β function does not suffer from the problems we described in the beginning of this section. Further, the ability to find optimal solutions with respect to β implies the ability to solve decision variants of manipulation. This is so because

$$perf_{sol(E)}(p) = perf_E(p) + \beta(E, s).$$

For a given input election E, $perf_E(p)$ is a constant. Solution sol ensures that p is a winner if and only if $perf_{sol(E)}(p)$ is nonnegative. Thus, p can become a winner if and only if

$$perf_E(p) + \max_{sol \in Sol(E,p)}(\beta(E, sol)) \geq 0,$$

where by $Sol(E, p)$ we mean the set of all valid solutions for our manipulation problem. Thus, maximizing β is a natural and useful goal for optimization variants of manipulation and other election problems.

Let α be a scoring protocol. An instance I of α-weighted-manipulation-max is a tuple (E, w, p) where $E = (C, V)$ is an election with candidate set C and weighted nonmanipulative voters V, $w = (w_1, \ldots, w_n)$ is a sequence of weights of the manipulative voters, and $p \in C$ is our preferred candidate. A solution is an assignment of preference lists to the manipulative voters. Our goal is to find a solution sol that maximizes $\beta(E, sol)$.

5.2.2 Results

Theorem 5.11 below is the main result of this section. Recall, from Section 2.4, that, given an instance I of a maximization problem, by $Opt(I)$ we mean the value of an optimal solution for I.

Theorem 5.11. *Let $\alpha = (\alpha_0, \ldots, \alpha_m)$ be a scoring protocol such that $\alpha_0 > \alpha_1$. There is an algorithm $\mathcal{A}$ that given a rational number ε, $0 < \varepsilon < 1$, and an instance $I = (E, w, p)$ of α-weighted-manipulation-max computes, in polynomial time in $|I|$ and $\frac{1}{\varepsilon}$, a solution sol such that $\beta(E, sol) \geq (1 - \varepsilon)Opt(I)$.*

It would be wonderful to have a single FPTAS that would work for all possible scoring protocols. However, we stress that the claim of Theorem 5.11 is that there is a *separate* algorithm for each *separate* scoring protocol $(\alpha_0, \dots, \alpha_m)$, where $\alpha_0 > \alpha_1$. In particular, each of the algorithms from Theorem 5.11 is tailored for a fixed number of candidates. Later on in this section we will show that it is unlikely (unless P = NP) that there exists a single general FPTAS for weighted manipulation in all possible scoring protocols.

We need some notation before we proceed with the proof of Theorem 5.11. Let $\alpha = (\alpha_0, \dots, \alpha_m)$ be a scoring protocol where $\alpha_0 > \alpha_1$ and let $C = \{p, c_1, \dots, c_m\}$ be a set of candidates. p is our preferred candidate whose performance we want to maximize. We implicitly assume that we have a set V of nonmanipulative voters, however in this discussion the only incarnation of the nonmanipulative voters is through the sequence s below. We let $w = (w_1, \dots, w_n)$ be the sequence of weights of the manipulators. Naturally, to maximize p's performance, each manipulator ranks p first. The complexity of α-weighted-manipulation-max comes from the difficulty in arranging the remainders of the manipulators' votes in such a way as to minimize the score of p's most dangerous competitor.

By $\mathcal{E}(C, w)$ we mean the set of all elections over the candidate set C with voter set containing exactly voters with weights $w_1, \dots, w_n$. Let $s = (s_1, \dots, s_m)$ be a sequence of nonnegative integers. Intuitively, the sequence s gives the scores that candidates c_1 through c_m receive from the nonmanipulative voters. By $S_\alpha(E, s)$ we mean $\max_{i \in \{1,\dots,m\}}\{score_E(c_i) + s_i\}$ and by $T_\alpha(w, s)$ we mean $\min_{E \in \mathcal{E}(C,w)} S_\alpha(E, s)$. Intuitively, function $T_\alpha(w, s)$ measures the smallest possible score that a highest-scoring candidate from $\{c_1, \dots, c_m\}$ can have after the manipulation. We now prove that for each scoring protocol α there is an FPTAS for T_α.

Lemma 5.12. *Let $\alpha = (\alpha_0, \dots, \alpha_m)$ be a scoring protocol and let $C = \{p, c_1, \dots, c_m\}$. There is an algorithm $\mathcal{T}$ that given a rational number ε, $0 < \varepsilon < 1$, a sequence $s = (s_1, \dots, s_m)$ of nonnegative integers and a sequence of manipulators weights $w = (w_1, \dots, w_n)$ computes an election $E \in \mathcal{E}(C, w)$ such that $S_\alpha(E, s) \leq (1+\varepsilon)T_\alpha(w, s)$. Algorithm $\mathcal{T}$ runs in polynomial time in n, m, and $\frac{1}{\varepsilon}$.*

Proof. Set $w_{\max} = \max\{w_1, \dots, w_n\}$ and set $K = \frac{\varepsilon w_{\max}}{n\alpha_1}$. Set $w' = (K\lceil\frac{w_1}{K}\rceil, \dots, K\lceil\frac{w_n}{K}\rceil)$.

It is possible to compute in polynomial time in n, m, and $\frac{1}{\varepsilon}$ an election $E' \in \mathcal{E}(C, w')$ such that $S_\alpha(E', s) = T_\alpha(w', s)$. (One can do so via a dynamic programming algorithm very similar to that devised in the proof of Theorem 5.8; we enforce that in our solution each voter ranks p first.) Let E be an election identical to E' only that appropriate voters have weights $w_1, \ldots, w_n$ instead of $w'_1, \ldots, w'_n$. Our algorithm outputs E.

It is easy to see that our algorithm can be made to work in polynomial time as required. Let us now show that the solution it produces satisfies the requirements regarding quality.

It is easy to see that $T_\alpha(w, s) \geq \alpha_1 w_{\max}$ and that $S_\alpha(E', s) \leq T_\alpha(w, s) + \alpha_1 nK$. The former is true because some candidate needs to get α_1 points from the manipulator with weight $w_{\max}$ and the second follows from the fact that for each i in $\{1, \ldots, n\}$ we have $w_i \leq w'_i < w_i + K$. For the same reason $S_\alpha(E, s) \leq S_\alpha(E', s)$.

Thus, $S_\alpha(E, s) \leq T_\alpha(w, s) + \alpha_1 nK = T_\alpha(w, s) + \varepsilon w_{\max}$. Since $T_\alpha(w, s) \geq \alpha_1 w_{\max}$, this yields that $S_\alpha(E, s) \leq (1 + \varepsilon) T_\alpha(w, s)$. (Note that, technically, this argument is only correct if $\alpha_1 \geq 1$ but, naturally, if $\alpha_1 = 0$ then the theorem is trivially satisfied.) This completes the proof. ❑

With Lemma 5.12 at hand we can prove Theorem 5.11.

Proof of Theorem 5.11. Our input is $I = (E, w, p)$, where $E = (C, V)$ is an election with candidate set $C = \{p, c_1 \ldots, c_m\}$ and collection V of nonmanipulative voters, $w = (w_1, \ldots, w_n)$ is a sequence of manipulators' weights, and p is our preferred candidate. Our goal is to find a solution *sol* (a collection of votes for the manipulators to cast) that (approximately) maximizes $\beta(E, sol)$.

Let $W = \sum_{i=1}^{n} w_i$ and let $w_{\max} = \max\{w_1, \ldots, w_n\}$. For each i in $\{1, \ldots, m\}$ let $s_i = score_E(c_i)$. We assume that the candidates $c_1, \ldots, c_m$ are listed in such an order that $s_1 \geq s_2 \geq \cdots \geq s_m$. Since $\alpha_0 > \alpha_1$, in every optimal solution each manipulator ranks p first and so, by definition of β, $Opt(I) = W\alpha_0 - (T_\alpha(w, s) - s_1)$. It would seem that computing approximately $T_\alpha(w, s)$ should be enough to get a good approximation of $Opt(I)$, but $T_\alpha(w, s)$ can be much bigger than $Opt(I)$. We have to, in some sense, reduce its value first.

Note that we can disregard all candidates c_j such that $s_1 - s_j > \alpha_1 W$. If there are k such candidates then the manipulators may simply rank them on the first k positions after p. For the sake of simplicity, we assume that there are no such candidates.

Let $s' = (s_1 - s_m, \ldots, s_m - s_m)$. It is easy to see that $Opt(I) = W\alpha_0 - (T_\alpha(w, s) - s_1) = W\alpha_0 - (T_\alpha(w, s') - s'_1)$. Additionally, via the above paragraph, we have that for each s'_i it holds that $s'_i \leq \alpha_1 W$. However, this means that $T_\alpha(w, s') \leq 2\alpha_1 W$. This is so because at worst the candidate whose score is the value of $T_\alpha(w, s')$ gets $\alpha_1 W$ points from s' and another $\alpha_1 W$ points from the manipulators.

Using algorithm $\mathcal{T}$ from Lemma 5.12, we fill-in the manipulators' votes to form an election $E' \in \mathcal{E}(C, w)$ such that all voters in E' rank p first and $T_\alpha(w, s') \leq S_\alpha(s', E') \leq (1 + \varepsilon')T_\alpha(w, s')$, where $\varepsilon' = \frac{1}{2\alpha_1}\varepsilon$. (Recall that in our setting α_1 is a constant.) Votes obtained in this way are the solution *sol* that our algorithm produces and we have $\beta(E, sol) = W\alpha_0 - (S_\alpha(s', E') - s'_1)$. Note that

$$\begin{aligned}
Opt(I) &= W\alpha_0 - (T_\alpha(w, s') - s'_1) \\
&\geq W\alpha_0 - (S_\alpha(s', E') - s'_1) \\
&\geq W\alpha_0 - ((1 + \varepsilon')T_\alpha(w, s') - s'_1) \\
&= W\alpha_0 - (T_\alpha(w, s') - s'_1) - \varepsilon' T_\alpha(w, s') \\
&= Opt(I) - \varepsilon' T_\alpha(w, s').
\end{aligned}$$

Since $Opt(I) \geq W$ (this is a consequence of the fact that $\alpha_0 > \alpha_1$), $T_\alpha(w, s') \leq 2\alpha_1 W$, and $\varepsilon' = \frac{1}{2\alpha_1}\varepsilon$, via the above calculations we have:

$$Opt(I) \geq W\alpha_0 - (S_\alpha(s', E') - s'_1) \geq (1 - \varepsilon)Opt(I).$$

Thus, $Opt(I) \geq \beta(E, sol) \geq (1 - \varepsilon)Opt(I)$. This completes the proof. ❑

Interestingly, we can use Theorem 5.11 to obtain results in spirit of those of Zuckerman, Procaccia, and Rosenschein (ZPR08), for the case of scoring protocols $\alpha = (\alpha_0, \ldots, \alpha_m)$ such that $\alpha_0 > \alpha_1$.

Theorem 5.13. *Let ε be a rational number, $0 < \varepsilon < 1$, and let $\alpha = (\alpha_0, \ldots, \alpha_m)$ be a scoring protocol such that $\alpha_0 > \alpha_1$. There is an algorithm that, given an instance $I = (E, w, p)$ of α-weighted-manipulation-max, where $w = (w_1, \ldots, w_n)$ is the* *sequence of manipulators' weights, has the property that if there is a manipulation that makes p a winner for instance I, the algorithm finds, in polynomial time in $|I|$ and $\frac{1}{\varepsilon}$, a successful manipulation for instance $I' = (E, (w_1, \dots, w_n, w_{n+1}), p)$, where $w_{n+1} = \lceil \varepsilon \max\{w_1, \dots, w_n\} \rceil$.*

Proof. The idea is to use the algorithm from the proof of Theorem 5.11 with a good enough value of ε and then supplement the solution with a single weight-w_{n+1} voter, who ranks p first.

Let α, I, and I' be as in the statement of the theorem and assume that there is a way to cast the manipulative voters' votes in I so that p becomes a winner of the election from that instance. That is, there is a solution sol'' such that

$$\mathit{perf}_{E(sol'')}(p) = \mathit{perf}_E(p) + \mathit{Opt}(I) \geq 0.$$

We now describe our algorithm. Let ε be a positive rational number as in the statement of the theorem. Let $w_{\max} = \max\{w_1, \dots, w_n\}$. Our algorithm works as follows.

1. Set $\varepsilon' = \frac{\varepsilon}{\alpha_0 n}$.
2. Run the algorithm from Theorem 5.11 on input (I, ε') to obtain a solution sol such that $\beta(E, sol) \geq (1 - \varepsilon')\mathit{Opt}(I)$.
3. Form a solution sol' for instance I' via adding to solution sol a single manipulator with weight $\lceil \varepsilon w_{\max} \rceil$ that ranks p first and that ranks all other candidates in an arbitrary order.

We claim that p is a winner of election $E(sol')$. To show that this is the case it suffices to show that $\mathit{perf}_{E(sol')}(p) \geq 0$. We will do so by showing that $\mathit{perf}_{E(sol')}(p) \geq \mathit{perf}_{E(sol'')}(p)$. Clearly, it holds that

$$\begin{aligned} \mathit{perf}_{E(sol)}(p) &= \mathit{perf}_E(p) + \beta(E, sol) \\ &\geq \mathit{perf}_E(p) + (1 - \varepsilon')\mathit{Opt}(I). \end{aligned}$$

Solution sol' is formed via adding to sol a manipulator with weight $\lceil \varepsilon w_{\max} \rceil$ who ranks p first and who ranks all the remaining candidates in an arbitrary order. Since $\alpha_0 > \alpha_1$,

$$\mathit{perf}_{E(sol')}(p) \geq \mathit{perf}_E(p) + (1 - \varepsilon')\mathit{Opt}(I) + \varepsilon w_{\max}.$$

Thus, to show that $\mathit{perf}_{E(sol')}(p) \geq \mathit{perf}_{E(sol'')}(p)$ it suffices to show that $(1-\varepsilon')\mathit{Opt}(I)+\varepsilon w_{\max} \geq \mathit{Opt}(I)$. It is easy to see that $\mathit{Opt}(I) \leq \alpha_0 n w_{\max}$ as this is the maximum value by which p's score can possibly grow due to including the manipulative voters. Since $\varepsilon' = \frac{\varepsilon}{\alpha_0 n}$, this means that

$$\begin{aligned}
(1-\varepsilon')\mathit{Opt}(I) + \varepsilon w_{\max} &= \mathit{Opt}(I) - \varepsilon' \mathit{Opt}(I) + \varepsilon w_{\max} \\
&= \mathit{Opt}(I) - \frac{\varepsilon}{\alpha_0 n}\mathit{Opt}(I) + \varepsilon w_{\max} \\
&\geq \mathit{Opt}(I) - \frac{\varepsilon}{\alpha_0 n}\alpha_0 n w_{\max} + \varepsilon w_{\max} \\
&= \mathit{Opt}(I) - \varepsilon w_{\max} + \varepsilon w_{\max} = \mathit{Opt}(I).
\end{aligned}$$

This completes the proof. ❑

Theorem 5.11 notwithstanding, we now show that for the case of an unbounded number of candidates there is no FPTAS for veto-weighted-manipulation-max, unless P $\neq$ NP.

Theorem 5.14. *If* P $\neq$ NP*, there is no FPTAS for veto-weighted-manipulation-max.*

To prove Theorem 5.14 it suffices to show that veto-weighted$_{\text{unary}}$-manipulation, that is, a version of veto-weighted-manipulation where weights are encoded in unary, is NP-complete. In unary-encoded variant of weighted manipulation in veto, and in each fixed scoring protocol, it holds that the maximum value of β function is polynomially bounded. Thus, if there were an FPTAS for veto-weighted-manipulation-max, then one could, via a sufficiently good approximation, solve veto-weighted$_{\text{unary}}$-manipulation exactly in polynomial time.

To show NP-hardness of veto-weighted$_{\text{unary}}$-manipulation we will reduce from Unary-3-Partition.

Name:	Unary-3-Partition
Given:	A unary encoded integer B and a multiset $A = \{a_1, \dots, a_{3m}\}$ of $3m$ positive integers such that (a) $\sum_{i=1}^{3m} a_i = mB$, and (b) for each $a_i \in A$ it holds that $\frac{B}{4} < a_i < \frac{B}{2}$.
Question:	Is there a partition A into m 3-element subsets $A_1, \dots, A_m$ such that elements in each A_i sum up to exactly B?

Unary-3-Partition is a well-known NP-complete problem (GJ79). Note that, since for each $a_i \in A$ it holds that $\frac{B}{4} < a_i < \frac{B}{2}$, any partition of A into m subsets $A_1, \dots, A_m$ that each sum up to B already implies that each of $A_1, \dots, A_m$ contains exactly 3 elements. Thus, we do not really need to require this in the problem definition.

Theorem 5.15. *veto-weighted*$_{\text{unary}}$*-manipulation is* NP*-complete.*

Proof. It is easy to see that veto-weighted$_{\text{unary}}$-manipulation is in NP. We now show that it is NP-hard via a reduction from Unary-3-Partition. Let (B, A) be our input instance of Unary-3-Partition where $A = \{a_1, \dots, a_{3m}\}$. We form an election $E = (C, V)$ where $C = \{p, c_1, \dots, c_m\}$ and where V contains a single voter with weight B and preference list

$$c_1 > c_2 > \cdots > c_m > p.$$

Additionally, there are $3m$ manipulative voters with weights $a_1, \dots, a_{3m}$. We claim that there is a way to cast the manipulators' votes to make p a winner if and only if $(B, A) \in$ Unary-3-Partition.

Not counting manipulators' votes, each candidate in C, except p, has B points, and p has 0 points. If there is a partition of A into $A_1, \dots, A_m$ such that each A_i sums up to B then there is a way to cast manipulators' votes to ensure p's victory. It is enough that for each A_i the manipulators corresponding to the elements of A_i vote

$$p > C - \{c_i\} > c_i.$$

Including such votes means that each candidate in the election is vetoed by candidates with weight exactly B and so each candidate, including p, is a winner.

For the converse, assume that there is a way to cast manipulators' votes to ensure p's victory. Without loss of generality we assume that each manipulator ranks p first. Thus, including manipulators' votes, p has mB points and, to ensure p's victory, each candidate $c_i \in C - \{p\}$ has to be vetoed by voters with vote weight at least B. However, since the joint vote weight of the manipulators is mB, there are m candidates in $C - \{p\}$, and each manipulator can veto only one candidate, it holds that each $c_i \in C - \{p\}$ is vetoed by manipulators with vote weight exactly B. It is easy to see that sets $A_1, \ldots, A_m$ such that each A_i contains elements of A corresponding to manipulators vetoing candidate c_i constitutes a 3-Partition of A. The reduction clearly works in polynomial time and the proof is complete. ❑

Thus, unless P = NP, Theorem 5.11 cannot be improved to give a single FPTAS for all scoring protocols.

5.3 Conclusions and Research Directions

In this chapter we have studied the complexity of bribery and manipulation in scoring protocols. In particular, we have shown that for most scoring protocols all standard variants of the weighted-bribery problem are NP-complete, but that for each scoring protocol unweighted variants of bribery (and variants where weights are encoded in unary) are in P. One of the explanations of this interesting behavior is that weighted voters maintain their individuality (their weights) even after we choose to bribe them, whereas priced voters are disassociated from their prices upon bribery. We have used our classification of scoring protocols with respect to bribery to obtain complexity results about manipulation in scoring protocols for the cases not covered by Hemaspaandra and Hemaspaandra (HH07), namely the cases of unweighted- and weighted$_{\text{unary}}$-manipulation.

In Section 5.2 we have considered the problem of approximately solving manipulation instances for scoring protocols. There is no agreed-upon, satisfactory framework for studying approximation algorithms for manipulation and we have developed a framework that we find very convincing. Our framework extends beyond manipulation and scoring protocols but here we have considered only this setting. Our main result of Section 5.2 is that for each scoring protocol $\alpha = (\alpha_1, \ldots, \alpha_m)$ such that $\alpha_1 > \alpha_2$, there is an FPTAS for α-weighted-manipulation. Thus, even though for most such scoring protocols weighted manipulation is NP-complete, via our result the problem can still be solved approximately in polynomial time.

This chapter opens some very natural research directions. In particular, it is very natural to ask if there are approximation algorithms (fully polynomial-time approximation schemes) for *all* scoring protocols, not only for the ones that assign more points to the top-ranked candidate than to any other one. That is, it would be very interesting to obtain a full classification of scoring protocols with respect to the existence of FPTASes for weighted manipulation. Similarly, it is natural to ask about approximation algorithms for bribery in scoring protocols. Even more generally, we are interested in obtaining results about approximation for bribery, manipulation, and control for other election systems. This line of research constitutes an alternative (and a complement) to the frequency-of-hardness approach (see the last paragraph of Section 1.2). We also point the reader to the work of Brelsford (Bre07) for some early work on approximation in the context of control problems.

6 Towards Perfect Resistance: Copeland Voting

So far, throughout this thesis we have explored many election systems and many settings for bribery and manipulation, but we have not yet seen an election system that would be resistant to *all* our standard flavors of bribery and manipulation. In this chapter we focus on Copeland$^\alpha$ voting and we show that Copeland$^\alpha$ *is* resistant to all of our standard variants of (constructive) bribery and to all standard variants of (constructive) manipulation.[1] Since Copeland$^\alpha$ is also resistant to essentially all standard types of (constructive) control[2] (FHHR07; FHHR08), then from the point of view of computational social choice, Copeland$^\alpha$ is one of the most promising election systems.

Copeland$^\alpha$ is currently the only system with a polynomial-time winner determination procedure that is known to possess such broad resistance to misuse. However, in Section 6.3 we show that in the irrational-voter model both Copeland^0 and Copeland^1 are vulnerable to microbribery. (Recall the discussion of approval microbribery in Section 4.2.3). On the other hand, the proofs of our hardness results for the standard type of bribery for the rational-voter model directly imply analogous hardness results for the irrational-voter model. The exact complexity of manipulation for the irrational-

[1]Our results for bribery hold for each rational α, $0 \leq \alpha \leq 1$, and our results on manipulation holds for each rational α, $0 < \alpha < \frac{1}{2}$. Faliszewski, Hemaspaandra, and Schnoor (FHS08) additionally show hardness of manipulation results for rational α such that $\frac{1}{2} < \alpha < 1$.

[2]Constructive control refers to scenarios where election organizers attempt to affect the result via such actions as adding/deleting/partitioning candidates/voters; we do not study control problems in this thesis but we stress that they are very important in computational social choice.

voter model for Copeland$^\alpha$ is currently unknown, but is being actively studied (Gre08). (Note that our hardness proof for manipulation in the rational-voter model relies on the rationality of the voters and thus does not imply irrational-voter model results.)

This chapter is organized as follows. In the next two sections we show resistance results for Copeland$^\alpha$ for the cases of bribery and manipulation. Then in Section 6.3, we show the forementioned vulnerability results for the case of microbribery of irrational voters. Finally, in the last section we present conclusions and some open research directions.

6.1 Bribery

Copeland$^\alpha$ is resistant to bribery for each rational α, $0 \leq \alpha \leq 1$, even if the voters are unweighted and do not have price tags. To prove this result we introduce a method of controlling the relative performances of certain voters in such a way that, if one sets up the reduction appropriately, the possibilities for successful bribery actions are sharply constrained. This technique proceeds by constructing bribery instances where the only briberies that could possibly ensure that our favorite candidate p is a winner involve only those voters who rank a group of special candidates above p. The remaining voters are used to create appropriate padding and structure within the election.

Theorem 6.1. *For each rational α, $0 \leq \alpha \leq 1$,* Copeland$^\alpha$*-bribery is* NP*-complete.*

Proof. Let α be an arbitrary rational α, $0 \leq \alpha \leq 1$. We give a reduction from X3C to Copeland$^\alpha$-bribery. Let $(B, \mathcal{S})$ be an instance of X3C, where $B = \{b_1, \ldots, b_{3k}\}$ and $\mathcal{S} = \{S_1, \ldots, S_n\}$. We assume that $n \geq k$, as otherwise there certainly is no cover of B with sets from $\mathcal{S}$. We form an election $E = (C, V)$, where $C = \{p, t, u, v\} \cup B$, and V contains two groups of voters, V' and V'', as specified below. For each S_i, V' contains one type (i) voter and one type (ii) voter:

$$\begin{array}{ll} \text{(i)} & t > u > v > S_i > p > B - S_i, \\ \text{(ii)} & \overrightarrow{B - S_i} > p > v > u > t > \overrightarrow{S_i}. \end{array}$$

V' also contains a single voter of type (iii) with preference list $B > p > u > v > t$. Since V' contains $2n+1$ voters, for each two candidates x, y in C it holds that $|\mathrm{vs}_{(C,V')}(x,y)|$ is odd and polynomially bounded in n. In particular, for each $b_i \in B$ it holds that $\mathrm{vs}_{(C,V')}(b_i, p) = 1$.

It is easy, if a little tedious, to see that for each two candidates x, y in $\{u, p, v, t\}$ there are two voters, v'_{xy} and v''_{xy}, such that

1. $\mathrm{vs}_{(C,\{v'_{xy}, v''_{xy}\})}(x,y) = 2$, and
2. for each pair of candidates $\{e, f\}$ such that $\{e, f\} \neq \{x, y\}$, $\mathrm{vs}_{(C,\{v'_{xy}, v''_{xy}\})}(e,f) = 0$,
3. for each $b_i \in B$ it holds that neither v'_{xy} nor v''_{xy} ranks all of $\{u, v, b_i\}$ ahead of p.

In essence, for each x, y in $\{u, p, v, t\}$ it suffices to partition C appropriately into C_1, C_2, and $\{x, y\}$, and use voters

$$\begin{aligned} v'_{x,y} &: \quad C_1 > x > y > C_2, \\ v''_{x,y} &: \quad \overrightarrow{C_2} > x > y > \overrightarrow{C_1}, \end{aligned}$$

where the candidates within C_1 and C_2 are ordered appropriately. Also, for each $x \in \{u, v, t\}$ voters,

$$\begin{aligned} v'_x &: \quad x > B > p > \{u, v, t\} - \{x\}, \\ v''_x &: \quad \overrightarrow{\{u, v, t\} - \{x\}} > p > x > \overrightarrow{B} \end{aligned}$$

have the property that for each $b_i \in B$ it holds that $\mathrm{vs}_{(C,\{v'_x, v''_x\})}(x, b_i) = 2$, and for each pair of candidate $\{e, f\}$, $\{e, f\} \neq \{x, b_j\}$ for each $b_j \in B$, $\mathrm{vs}_{(C,\{v'_x, v''_x\})}(e,f) = 0$.

We form voter set V'' via including pairs of voters $v'_{x,y}$ and $v''_{x,y}$ (for obvious choices of $x, y \in \{u, v, p, t\}$) and pairs of voters v'_x and v''_x (for obvious choices of $x \in \{u, v, t\}$) such that in (C, V) we obtain the following:

1. $\mathrm{vs}_E(u,p) = \mathrm{vs}_E(v,p) = 2k - 1$,
2. $\mathrm{vs}_E(u,v) = 2k + 1$,
3. $\mathrm{vs}_E(t,u) = \mathrm{vs}_E(t,p) = 2k + 1$,
4. $\mathrm{vs}_E(v,t) = 2k + 1$,

5. for each $b_i \in B$, $\text{vs}(u, b_i) = \text{vs}(v, b_i) = \text{vs}(t, b_i) \geq 2k+1$, and

6. for each $b_i \in B$, $\text{vs}(b_i, p) = 1$.

It is easy to see that appropriate V'' can be computed in polynomial time. The above results of head-to-head contests yield the following scores.

1. $\mathit{score}_E^{\alpha}(u) = \mathit{score}_E^{\alpha}(v) = \mathit{score}_E^{\alpha}(t) = 3k+2$,

2. $\mathit{score}_E^{\alpha}(p) = 0$, and

3. for each $b_i \in B$, $\text{Copeland}^{\alpha}{}_E(b_i) \leq 3k$.

The results of head-to-head contests that are won by $2k+1$ points or more cannot be changed (from one candidate winning to the other one winning) via briberies of at most k voters. Thus, the only head-to-head contests that may change their results due to a bribery of at most k voters are those between u and p, between v and p, and between p and each $b_i \in B$. In particular, a bribery of up to k voters cannot affect in any way the score of t, and, in effect, a bribery of up to k voters cannot increase scores of u and v. Also, neither of the candidates $b_i \in B$ can obtain a score higher than $3k$ via a bribery of up to k voters.

We claim that p may become a winner of E via a bribery of at most k voters if and only if $(B, \mathcal{S}) \in \text{X3C}$. If $\mathcal{S}$ contains an exact cover of B then p can become a winner via bribing the k type (i) voters in V' that correspond to a cover. After bribing these voters to rank p first, p wins his or her head-to-head contests with u, v and each candidate in B. This gives p score $3k+2$. Via previous paragraph, this means that p is a winner.

Now, for the other direction, assume that there is a bribery of k voters such that after this bribery p is a winner. Irrespective of which k voters are bribed, t's score is $3k+2$. Thus, any bribery of k voters that ensures that p is a winner must guarantee that p wins his or her head-to-head contests with both u and v, and with each member of B. Since p loses his or hers head-to-head contests with each of u and v by exactly $2k-1$ votes, a successful bribery involves bribing k voters that each rank both u and v ahead of p. In addition, each successful bribery of at most k voters ensures that p wins his or hers head-to-head contests with each $b_i \in B$, and so each voter bribed in such a successful bribery ranks at least one member of B ahead of p. By construction of V the only such voters are type (i) voters. Via a simple counting argument it is easy to see that it is possible to bribe k type (i) voters so that p wins head-to-head contests with u, v, and each member of B only if it is possible to pick k type (i) voters that correspond to a cover of B by sets in $\mathcal{S}$. The proof is complete. ❑

Note that the above proof does not rely upon the fact that the voters are rational. In our reduction we can allow the voters to be irrational and form bribery instances simply by deriving the voters' preference tables from the voters' given preference lists. It is easy to see that the proof remains valid after this change; in our argument we assume that each bribed voter, after the bribery, prefers p to all other candidates but we do not make any further assumptions (and, in particular, we do not use linearity of the preferences). Thus we have the following corollary.

Corollary 6.2. *For each rational number α, $0 \leq \alpha \leq 1$, bribery for* Copeland$^{\alpha}$ *in the irrational-voter model is* NP*-complete.*

Via Theorem 6.1 and Corollary 6.2 we have: For each rational α, $0 \leq \alpha \leq 1$, Copeland$^{\alpha}$ possesses broad—essentially perfect—resistance to bribery, regardless of whether we are interested in rational or irrational voters. Naturally, this hardness result directly translates to the more involved cases of weighted voters, priced voters, and weighted-and-priced voters. However, for the case of \$bribery and weighted-\$bribery we can obtain an even stronger result. In the next section we show that Copeland$^{\alpha}$, for the case of rational α such that $0 < \alpha < \frac{1}{2}$, is resistant to manipulation, even if there are only two manipulators (and, in fact, this result holds for rational α's such that $\frac{1}{2} < \alpha < 1$ as well; see (FHS08)). Thus, via Theorem 3.4, we conclude that Copeland$^{\alpha}$ (for rational α such that $0 < \alpha < 1$, $\alpha \neq \frac{1}{2}$) is resistant to \$bribery and weighted-\$bribery even if the budget is so limited that at most two voters can be bribed.

6.2 Manipulation

We now turn to the issue of manipulation in Copeland$^\alpha$ voting. Our main result here is that Copeland$^\alpha$, for the case of rational α such that $0 < \alpha < \frac{1}{2}$, is resistant to manipulation, even if there are only two manipulators.[3]

Theorem 6.3. *Let α be a rational number such that $0 < \alpha < \frac{1}{2}$.* Copeland$^\alpha$-manipulation *is* NP*-complete.*

Our proof involves building fairly complicated instances of elections and we first provide some results that simplify crafting such instances.

6.2.1 Constructing Manipulation Instances

Each election $E = (C, V)$ induces a directed graph $G(E)$ with edges labeled with non-negative integers. Vertices of $G(E)$ are exactly the candidates of E and edges correspond to the results of head-to-head contests between the candidates. That is, for each two distinct candidates $c_i, c_j \in C$ we have an edge in $G(E)$ from c_i to c_j with label k if and only if $\mathrm{vs}_E(c_i, c_j) = k$ and $k > 0$. The following lemma, due to McGarvey (McG53) (see also (Ste59)), says that each directed, antisymmetric graph with edges labeled by nonnegative even integers is induced by an election.

Lemma 6.4 ((McG53))**.** *For each antisymmetric directed graph G with edges labeled with nonnegative even integers, there exists an election E such that $G = G(E)$, and E can be computed in polynomial time in the size of G and the largest label.*

Proof. For the sake of completeness, we give the algorithm. Let G be an antisymmetric directed graph. The algorithm computes the election $E = (C, V)$, where $C = V(G)$ and for each edge (a, b) in G with label k there are exactly k voters, $\frac{k}{2}$ with preference list $a > b > C - \{a, b\}$ and $\frac{k}{2}$ with preference list $\overrightarrow{C - \{a, b\}} > a > b$. Since G is antisymmetric and k is even, it is easy to see that $G = G(E)$. ❑

[3]Single-voter manipulation is easy for Copeland$^\alpha$. This follows easily from (BTT89a).

We will sometimes construct complicated elections by combining simpler ones. Whenever we speak of *combining* two elections, say $E_1 = (C_1, V_1)$ and $E_2 = (C_2, V_2)$, we mean building (via the algorithm from Lemma 6.4) an election $E = (C, V)$ whose election graph is a disjoint union of the election graphs of E_1 and E_2 with, possibly, added edges between the vertices of $G(E_1)$ and $G(E_2)$ (explicitly stating which edges, if any, are added). In particular, we will often want to add some padding candidates to an election, without affecting the original election significantly. In order to do so, we will typically combine our main election with one of the following "padding" ones.

Lemma 6.5. *Let α be a rational number such that $0 \leq \alpha \leq 1$. For each positive integer n, there is a polynomial-time (in n) computable election $\mathrm{Pad}_n = (C, V)$ such that $\|C\| = 2n + 1$ and for each candidate $c_i \in C$ it holds that $score^{\alpha}_{\mathrm{Pad}_n}(c) = n$.*

Proof. Fix a positive integer n. By Lemma 6.4 it is enough to construct (in polynomial time in n) a directed, antisymmetric graph G with $2n+1$ vertices, each with its indegree and outdegree equal to n. We set G's vertex set to be $\{0, 1, \ldots, 2n\}$ and we put an edge from vertex i to vertex j ($i \neq j$) if and only if $(j - i) \bmod (2n+1) \leq n$. As a result there is exactly one directed edge between every two distinct vertices and for each vertex i we have edges going out from i to exactly the vertices $(i+1) \bmod (2n+1), (i+2) \bmod (2n+1), \ldots, (i+n) \bmod (2n+1)$. Thus, both the indegree and the outdegree of each vertex is equal to n and the proof is complete. ❑

The next lemma allows us to easily construct elections without specifying the results of all head-to-head contests, but via giving the scores and results of head-to-head contests for relevant candidates only.

Lemma 6.6. *Let $E = (C, V)$ be an election where $C = \{c_1, \ldots, c_{n'}\}$, let α be a rational number such that $0 \leq \alpha \leq 1$, and let $n \geq n'$ be an integer. For each candidate c_i we denote the number of head-to-head ties of c_i in E by t_i. Let q be a positive integer and let $k_1, \ldots, k_{n'}$ be a sequence of n' nonnegative integers such that for each k_i we have $0 \leq k_i \leq n^q$. There is an algorithm that in polynomial time in n outputs an election*

$E' = (C', V')$ *such that:*

1. $C' = C \cup D$, *where* $D = \{d_1, \dots, d_{2n'n^q}\}$,

2. E' *restricted to* C *is* E *(that is,* $G(E')$ *restricted to* C *is* $G(E)$*),*

3. *the only ties in head-to-head contests in* E' *are between candidates in* C,

4. *for each* i, $1 \leq i \leq n'$, $score^{\alpha}_{E'}(c_i) = 2n'n^q - k_i + t_i\alpha$, *and*

5. *for each* i, $1 \leq i \leq 2n'n^q$, $score^{\alpha}_{E'}(d_i) \leq n'n^q + 1$.

Proof. We build E' via combining E with a padding election F (see Lemma 6.5 and the paragraph just before it). $F = (D, W)$, where $D = \{d_1, \dots, d_{2n'n^q}\}$, is essentially the election $\mathrm{Pad}_{n'n^q}$ with one arbitrary candidate removed. We partition the candidates in D into n' groups, $D_1, \dots, D_{n'}$, each with exactly $2n^q$ candidates and we set the results of head-to-head contests between each $c_i \in C$ and the candidates in D according to the following scheme. For each $j \in \{1, \dots, n'\}$ such that $i \neq j$, c_i defeats all members of D_j and c_i defeats exactly as many candidates in D_i (and loses to all the remaining ones) as needed to ensure that

$$score^{\alpha}_{E'}(c_i) = 2n'n^q - k_i + t_i\alpha.$$

It is easy to see that this is possible: c_i's score in $(C' - D_i, V')$ is $2n'n^q - 2n^q + k' + t_i\alpha$ for some k' such that $0 \leq k' \leq n' - t_i$. There are $2n^q$ candidates in D_i and so c_i certainly can reach any score of the form $2n'n^q - k + t_i\alpha$, where k is an integer between 0 and n^q, via defeating in head-to-head contests an appropriate number of candidates in D_i and losing to all the remaining ones.

Finally, since F is $\mathrm{Pad}_{n'n^q}$ with one candidate removed, each d_i gets at most $n'n^q$ points from defeating other members of D and at most one point from possibly defeating some member of C. Thus, for each $d_i \in D$, it holds that $score^{\alpha}_{E'}(d_i) \leq n'n^q + 1$. This completes the proof. ❑

Note that in the proof of Lemma 6.6 we never introduce head-to-head contest ties other than those already present in the election E.

We conclude with the following observation: Let $E = (C, V)$ be an election and let c_i and c_j be two candidates. We can add two voters, v and v', one with preference order $c_i > c_j > C - \{c_i, c_j\}$, and the other with preference order $\overrightarrow{C - \{c_i, c_j\}} > c_i > c_j$, so that we do not change the result of any of the head-to-head contests except the one between c_i and c_j, where we give c_i two votes of advantage. Thus, we can build elections using Lemma 6.6 and then amplify the results of specific head-to-head contests as we please.

Let us now turn to the issue of building manipulation instances needed in the proof of Theorem 6.3. The reduction that we construct in that proof accepts as input an X3C instance I and outputs a manipulation instance with an election $E = (C, V)$, preferred candidate $p \in C$, and two manipulators, v and v'. In the proof we specify this instance by giving a collection of candidates $c_1, \ldots, c_n$, their scores relative to the score of p, and the results of head-to-head contests (among $c_1, \ldots, c_m$) that the manipulators can affect. (Each of these contests is won/lost by exactly two votes; we refer to these contests as *flexible* and to all the other ones as *nonflexible.*) In the proof we assume that the manipulators cannot (would not) affect results of any head-to-head contests other than these flexible ones. The remainder of this section is devoted to showing that thus specified instances of manipulation can in fact be constructed in polynomial time. On the surface, it seems that one can build such instances by simply invoking Lemma 6.6, but this is not the case. Our reduction relies on candidates $c_1, \ldots, c_n$ having scores of the form $x + \alpha y$, where y is not necessarily 0. The natural way of constructing elections where candidates have such scores involves introducing ties in head-to-head contests. However, such head-to-head contests could then be affected by the manipulators, and the proof relies on the fact that this is not possible. Thus, in what follows, we describe an alternative way of obtaining scores of this form.

Since the manipulators' goal is to ensure p's victory, we assume, without loss of generality, that the manipulators always rank p first.

We design E to have an even number of voters and to have all head-to-head contests won/lost by more than two votes unless specified otherwise. In our construction of E we ensure that, not counting the manipulators' votes, p obtains K points, where K is some fairly large integer. p receives each of these K points by winning a head-to-head contest. p loses all remaining head-to-head contests—except for one—by more than 2 votes. This one singled-out head-to-head contest is between p and a special candidate t. p loses this contest by *exactly* 2 votes. Since both manipulators rank p first, including their votes we have that p's final, "postmanipulation," Copeland$^\alpha$ score in E is $\ell = K + \alpha$. We ensure that t has Copeland$^\alpha$ score lower than K and we never use candidate t for purposes other than this in our construction. (Of course, we still have to assign results of head-to-head contests between t and all other candidates, but this can be done automatically via an invocation of Lemma 6.6.)

For each candidate c_i, $1 \leq i \leq n$, let f_i and t_i be two nonnegative integers such that our reduction requires candidate c_i to have prior-to-manipulation Copeland$^\alpha$ score $K + \alpha t_i \pm f_i$. Implementing the integral part of the score would be an easy invocation of Lemma 6.6. But how do we implement α parts of the scores?

Let $T = \sum_{i=1}^{n} t_i$. In our reduction T is polynomially bounded in the size of I. We introduce T candidates $e_1, \ldots, e_T$ and we require that their Copeland$^\alpha$ scores, not counting the manipulators' votes, are exactly $K+1$. We also set that for each c_i, exactly t_i distinct candidates from the set $\{e_1, \ldots, e_T\}$ win their head-to-head contests with c_i, each by exactly two votes. We build our election by invoking Lemma 6.6 for candidates $\{p, t, c_1, \ldots, c_n, e_1, \ldots, e_T\}$, requiring p to have K points, t to have less than K points, each c_i to have exactly f_i points, each e_i to have $K + 1$ points, and with head-to-head contests set arbitrarily except those mentioned explicitly above (including the flexible head-to-head contests among $c_1, \ldots, c_n$).

How does this construction work? Before we "begin the manipulation" it does not work at all. However, it is easy to see that if p is to become a winner then both manipulators, v and v', have to guarantee that each candidate e_j in $\{e_1, \ldots, e_T\}$ ties the head-to-head contest with the candidate c_i that e_j used to defeat by 2 votes, thus giving each c_i the additional αt_i points. The reason is that p can at best have Copeland$^\alpha$ score $\ell = K + \alpha$ and, had v and v' not ensured all the ties we mention then at least one of $e_1, \ldots, e_T$ would have Copeland$^\alpha$ score $K + 1 > K + \alpha$ (recall that $0 < \alpha < \frac{1}{2}$). p would not be a winner. v and v' can ensure that all these ties happen by listing each of $c_1, \ldots, c_n$ before any of $e_1, \ldots, e_T$ in their votes.

Thus, we can assume that we can specify the α parts of the scores of c_i's. In our specifications of Copeland$^\alpha$ scores for candidates we can also use expressions of the form $f_i - \alpha t_i$ because for each rational α in $(0, 1)$ there are two nonnegative integer constants, s_1 and s_2, such that $\alpha s_1 = s_2 - \alpha$, which we can use to express this subtraction.

6.2.2 The Proof

This section constitutes the proof of Theorem 6.3. We show, via a reduction from X3C, that for each rational α, $0 < \alpha < \frac{1}{2}$, Copeland$^\alpha$-manipulation is NP-hard.

Let $I = (B, \mathcal{S})$ be an instance of X3C, where $B = \{b_1, \ldots, b_{3k}\}$ and $\mathcal{S} = \{S_1, \ldots, S_n\}$ is some family of 3-element subsets of B. We describe an election E where two manipulative voters, v and v', can ensure a distinguished candidate p's victory if and only if I is a *yes*-instance of X3C. Note that, following the discussion in Section 6.2.1, we will only describe significant candidates and omit the padding ones. Similarly, we will express scores that our candidates have before manipulators' votes are counted in the form $\ell + f + \alpha t$, where ℓ is the (essentially fixed) number of points that p obtains. From now on when describing E we will use the word "candidates" to refer only to the significant candidates, but one should keep in mind that, of course, the padding ones are there as well.

Given $(B, \mathcal{S})$, we build our election E to have the following candidates. (Below we also give their prior-to-manipulation Copeland$^\alpha$ scores and their flexible head-to-head contests.)

p**.** The distinguished candidate whose victory we want to ensure. By the discussion below Theorem 6.3, after manipulation p has exactly ℓ Copeland$^\alpha$ points.

$b_1, \ldots, b_{3k}$**.** For each $i \in \{1, \ldots, 3k\}$ we have a single candidate b_i with Copeland$^\alpha$ score $\ell - \alpha$.

$S_1, \ldots, S_n, z_{1_1}, \ldots z_{n_3}$**.** For each set S_i we have a single candidate S_i with score $\ell + 3 - 3\alpha$. Each S_i defeats in their head-to-head contests exactly those b_j's that are members of S_i (these victories are by 2 votes each).

c**.** The *counter* candidate; has Copeland$^\alpha$ score $\ell - (n-k)\alpha$.

$z_{1_1}, \ldots, z_{n_3}$**.** Candidates z_{i_1}, z_{i_2}, and z_{i_3}, $i \in \{1, \ldots, n\}$ are responsible for implementing a certain consistency gadget. For each $i \in \{1, \ldots, n\}$ we have that z_{i_1} wins by 2 votes the head-to-head contest with z_{i_2}, z_{i_2} wins by 2 votes the head-to-head contest with z_{i_3}, and z_{i_3} wins by 2 votes the head-to-head contest with c. Also, each S_i defeats, by 2 votes, each of the candidates z_{i_1}, z_{i_2}, and z_{i_3}. The form of Copeland$^\alpha$ scores of candidates z_{i_t} depends on α and we specify it later.

Aside from head-to-head contests mentioned above, all other head-to-head contests are either won or lost by more than 2 votes.

For each $i \in \{1, \ldots, n\}$, S_i refers to both a candidate in E and a set in $\mathcal{S}$. Similarly, for each $i \in \{1, \ldots, n\}$, b_{i_1}, b_{i_2}, and b_{i_3} refer to candidates in E as well as to the elements of $S_i \in \mathcal{S}$.

Each candidate S_i, $i \in \{1, \ldots, n\}$ has a surplus of $3-3\alpha$ Copeland$^\alpha$ points. To ensure p's victory, the manipulators have to case such votes that remove this surplus. For each S_i we can do so via enforcing that S_i ties with at least three of $z_{i_1}, z_{i_2}, z_{i_3}, b_{i_1}, b_{i_2}, b_{i_3}$. Later we show how to specify scores of candidates $z_{i_1}, z_{i_2}, z_{i_3}$, $i \in \{1, \ldots, n\}$ in such a way that in every manipulation that guarantees p's victory, if S_i ties with at least one of z_{i_1}, z_{i_2}, or z_{i_3} then c ties with some candidate that he or she used to lose to. Thus, c's score increases by α for each such S_i. We now show that this implies that any manipulation that ensures p's victory has to guarantee that each S_i either ties with all three of $b_{i_1}, b_{i_2}, b_{i_3}$ or with neither of them.

Let us assume that there is a way for v and v' to cast their votes in such a way that p is a winner. This means that after the manipulation all other candidates have scores at most ℓ. For each $j \in \{0,1,2,3\}$, let K_j be the number of candidates S_i that, including votes of v and v', tie with exactly j of $b_{i_1}, b_{i_2}, b_{i_3}$. Since there are exactly n candidates S_i, we have that

$$K_0 + K_1 + K_2 + K_3 = n. \tag{6.1}$$

Each S_i that is not accounted for in K_3 has to tie with at least one z_{i_t}, $t \in \{1,2,3\}$, and for each such S_i the counter candidate c gets extra α points. Thus,

$$K_0 + K_1 + K_2 \le n - k, \tag{6.2}$$

as $(n-k)\alpha$ is the largest number of points c can accept without having his or her score over ℓ (recall our gadget connecting z_{i_t}'s and c). Finally, since there are exactly $3k$ candidates $b_1, \dots, b_{3k}$, and each of them can tie with at most one S_i, we have

$$3K_3 + 2K_2 + 1K_1 \le 3k. \tag{6.3}$$

If we sidewise add to it inequality (6.2) multiplied by 3 then we obtain

$$3K_0 + 3K_1 + 3K_2 + 3K_3 + 2K_2 + 1K_1 \le 3n. \tag{6.4}$$

Since, via (6.1),

$$3K_0 + 3K_1 + 3K_2 + 3K_3 = 3n, \tag{6.5}$$

we have that $2K_2 + K_1 \le 0$. Since K_1 and K_2 are nonnegative integers, it implies that K_1 and K_2 are 0; after the manipulation each candidate S_i either ties with each of b_{i_1}, b_{i_2}, b_{i_3} or with none of them.

Thus, if p is to be a winner, at most k of candidates S_i can tie with candidates corresponding to the members of S_i (because $3K_3 + 2K_2 + K_1 \le 3k$ and both K_2 and K_1 are 0). Since, at most $n-k$ of S_i's can tie with their associated candidates $z_{i_1}, z_{i_2}, z_{i_3}$, it is easy to see that those S_i's that tie with the corresponding candidates $b_{i_1}, b_{i_2}, b_{i_3}$ constitute exactly an exact-3-cover of B. To finish this direction of the proof it remains to show that for any manipulators' votes that ensure p's victory it really is the case that for each S_i that ties with at least one of $z_{i_1}, z_{i_2}, z_{i_3}$ (including manipulators' votes) candidate c's score increases by α.

We set the scores of candidates $z_{1_1}, \dots, z_{n_3}$ depending on the value of α. We first handle the case when $\frac{1}{3} \le \alpha < \frac{1}{2}$. In this case, we declare that each of z_{i_t}, $i \in \{1, \dots, n\}$, $t \in \{1,2,3\}$, has score exactly $\ell + 1 - 3\alpha$. (Note that since $\alpha \ge \frac{1}{3}$, $1 - 3\alpha \le 0$.) It is easy to see that if any of z_{i_t} obtains extra α (or more) points from tieing either with S_i or $z_{i_{t-1}}$ (provided $t > 0$ for the latter) then we need to ensure that this z_{i_t} also "unloads" these extra points somewhere. The only way to decrease z_{i_t}'s score is via ensuring that he or she ties with $z_{i_{t+1}}$ (or c, if $t = 3$). Since $\alpha < \frac{1}{2}$, the amount of points z_{i_t} loses this way balances all the points z_{i_t} might obtain due to manipulation and ensures that his or her score is at most ℓ. Also, due to z_{i_t} tieing with $z_{i_{t+1}}$, $t \in \{1, 2\}$, $z_{i_{t+1}}$ obtains extra α points he or she needs to unload. This way the effect of S_i tieing with either one of z_{i_t}'s ($t \in \{1, 2, 3\}$) propagates to eventually increasing c's score by α. The reader can easily verify that if each of $z_{i_1}, z_{i_2}, z_{i_3}$ ties with S_i, z_{i_1} ties with z_{i_2}, z_{i_2} ties with z_{i_3} and z_{i_3} ties with c then each of $z_{i_1}, z_{i_2}, z_{i_3}$ has Copeland$^\alpha$ score at most ℓ. Such tieing can be implemented by manipulators v and v' if in both their votes they prefer c to z_{i_3} to z_{i_2} to z_{i_1} to S_i.

For the case of rational α such that $0 < \alpha \leq \frac{1}{3}$ it is easy to see that the same arguments work provided that each candidate z_{i_t}, $i \in \{1, \dots, n\}$, $t \in \{1, 2, 3\}$ starts with Copeland$^\alpha$ score equal to ℓ.

We now show that if I is a *yes*-instance of X3C then p can become a winner of our election. Let $\mathcal{S}_C$ be a set of all candidates S_i that correspond to some exact-3-cover of B and let Z_C be the set of their corresponding z_{i_t} candidates. Let Z be the set of all the z_{i_t} candidates in the election. It is easy to check that voters v and v' can ensure p's victory via casting votes as follows.

$$\begin{aligned} v &: \; p > c > Z_C > \mathcal{S}_C > B > \mathcal{S} - \mathcal{S}_C > Z - Z_C > \cdots \\ v' &: \; p > \mathcal{S} - \mathcal{S}_C > Z - Z_C > c > Z_C > \mathcal{S}_C > B > \cdots \end{aligned}$$

The ellipsis means the padding candidates, listed in arbitrary order. It is easy to verify that with these votes all candidates end up with Copeland$^\alpha$ score of at most ℓ and that p gets exactly ℓ points, becoming a winner. The proof is complete.

Interestingly, by replacing our consistency gadgets with other constructions, it is possible to adapt the above proof to work with rational α's between $\frac{3}{4}$ and 1. However, the proof of Faliszewski, Hemaspaandra, and Schnoor (FHS08) for the case of α strictly between $\frac{1}{2}$ and 1 employs a somewhat different approach. The case of $\alpha = \frac{1}{2}$ remains open.

6.3 Microbribery

In this section we explore microbribery of irrational voters in Copeland$^\alpha$. In standard bribery problems for irrational voters we would be asking whether it is possible to ensure that a designated candidate p is a winner by modifying the preference tables of at most k voters. In microbribery we have to pay separately for each preference-table entry flip and, thus, we pay more the more we alter a vote. Copeland$^\alpha$ microbribery problems are very similar in flavor to the approval microbribery problems that we studied in Section 4.2.3, but the proofs for Copeland$^\alpha$ seem to be much more involved than their counterparts for approval voting. The reason is that Copeland$^\alpha$ elections allow for subtle and complicated interactions between the candidates' scores.

For each rational α, $0 \leq \alpha \leq 1$, we define Copeland$^\alpha$-microbribery to be the following decision problem.

Name: Copeland$^\alpha$-microbribery.

Given: An election $E = (C, V)$, where each voter has a preference table over the candidates in C, a distinguished candidate $p \in C$, and a nonnegative integer k.

Question: Is it possible, by flipping at most k entries in the preference tables of voters in V, to ensure that p is a Copeland$^\alpha$ winner of the resulting election?

Throughout the remainder of this section we will use the term *microbribe* to refer to flipping an entry in a preference table, and we will use the term *microbribery* to refer to bribing possibly irrational voters via microbribes. Recall that by "irrational voters" we simply mean that they are allowed to have, but not that they must have, irrational preferences.

Let E be an election with candidate set $C = \{c_1, c_2, \ldots, c_m\}$ and voter collection $V = (v_1, v_2, \ldots, v_n)$. We define two functions, $\mathit{wincost}_E$ and $\mathit{tiecost}_E$, that describe the costs of ensuring a victory or a tie of a given candidate in a particular head-to-head contest.

Definition 6.7. *Let $E = (C, V)$ be an election and let c_i and c_j be two distinct candidates in C.*

1. *By wincost$_E(c_i, c_j)$ we mean the minimum number of microbribes that ensures that c_i defeats c_j in their head-to-head contest. If c_i already wins this contest then wincost$_E(c_i, c_j) = 0$.*
2. *By tiecost$_E(c_i, c_j)$ we mean the minimum number of microbribes that ensures that c_i ties with c_j in their head-to-head contest, or ∞ if E has an odd number of voters and thus ties are impossible.*

The next theorem is our main result of this section.

Theorem 6.8. *For $\alpha \in \{0, 1\}$,* Copeland$^\alpha$*-microbribery is in* P*.*

We prove Theorem 6.8 via Lemmas 6.9 through 6.13 below, which cover three cases: (a) an odd number of voters, where all Copeland$^\alpha$ elections with $0 \leq \alpha \leq 1$ are identical due to the lack of ties, (b) Copeland^1 with an even number of voters, and (c) Copeland^0 with an even number of voters.

Lemma 6.9. *For each rational α with $0 \leq \alpha \leq 1$, there is a polynomial-time algorithm that solves the constructive microbribery problem for* Copeland$^\alpha$ *elections with an odd number of voters.*

Proof. Our input is a nonnegative integer k (the budget) and an election $E = (C, V)$, where the candidate set C is $\{c_0, c_1, \ldots, c_m\}$, the number of voters is odd, and $p = c_0$ is the candidate whose victory we want to ensure via at most k microbribes. Note that, since it is sometimes convenient to be able to speak of p and all other candidates uniformly, we use p and c_0 to refer to the same candidate interchangeably. As the number of voters is odd, ties never occur and the particular value of α used does not affect Copeland$^\alpha$ score of any candidate. Fix an arbitrary such α.

We give a polynomial-time algorithm for the constructive microbribery problem. A high-level overview is that we try to find a threshold value T such that there is a

Edge	Parameters
$e = (s, c_i)$, where $c_i \in C$	$c(e) = score^{\alpha}_{E}(c_i)$ $a(e) = 0$
$e = (c_i, c_j)$, where $c_i, c_j \in C$ and $\mathrm{vs}_E(c_i, c_j) > 0$	$c(e) = 1$ $a(e) = wincost_E(c_j, c_i)$
$e = (c_0, t)$	$c(e) = T$ $a(e) = 0$
$e = (c_i, t)$, where $i > 0$ and $c_i \in C$	$c(e) = T$ $a(e) = B$
every other edge e	$c(e) = 0$ $a(e) = 0$

Figure 6.1: Edge capacities and costs for min-cost-flow instance $I(T)$, built from election E.

microbribery of cost at most k that transforms E into E' such that (a) p has $score^{\alpha}_{E'}$ exactly T, and (b) every other candidate has $score^{\alpha}_{E'}$ at most T.

Let B be a number that is greater than the cost of any possible microbribery within E (e.g., $B = \|V\| \cdot \|C\|^2 + 1$). For each possible threshold T, we consider a min-cost-flow instance $I(T)$ with node set $K = C \cup \{s, t\}$, where s is the source and t is the sink, the edge capacities and costs are specified in Figure 6.1, and the target flow value is

$$F = \sum_{c_i \in C} score^{\alpha}_{E}(c_i) = \frac{\|C\|(\|C\| - 1)}{2}.$$

Note that with an odd number of voters, constructive microbribery in Copeland$^{\alpha}$ simply requires us to choose for which pairs of distinct candidates we want to flip the outcome of their head-to-head contest in order to ensure p's victory. Thus it is sufficient to represent a microbribery M as a collection of pairs (c_i, c_j) of distinct candidates for whom we need to flip the result of their head-to-head contest from c_i winning to c_j

winning. Clearly, given such a collection M, the cheapest way to implement it costs

$$\sum_{(c_i,c_j)\in M} \mathit{wincost}_E(c_j, c_i).$$

A crucial observation for our algorithm is that we can directly translate flows to microbriberies using the following interpretation. Let f be a flow (as per Definition 2.4 with all edge flows being integers) of value F within instance $I(T)$. The units of flow that travel through the network correspond to Copeland$^\alpha$ points. For each $c_i \in C$, we interpret the amount of flow that goes directly from s to c_i as the number of Copeland$^\alpha$ points that c_i has before any microbribery is attempted,[4] and the amount of flow that goes directly from c_i to t as the number of Copeland$^\alpha$ points that c_i has after the microbribery (defined by the flow). The units of flow that travel between distinct c_i's (i.e., through edges of the form (c_i, c_j), $i \neq j$) correspond to the microbribes exerted: A unit of flow traveling from node c_i to c_j corresponds to changing the result of the head-to-head contest between c_i and c_j from c_i winning to c_j winning. In this case, the Copeland$^\alpha$ point moves from c_i to c_j and the cost of the flow increases by $a(c_i, c_j) = \mathit{wincost}(c_j, c_i)$, exactly the minimum cost of a microbribery that flips this contest's result. Let M_f be the microbribery defined, as just described, by flow f. It is easy to see that

$$\mathit{flowcost}(f) = B \cdot (F - f(c_0, t)) + \sum_{(c_i,c_j)\in M_f} \mathit{wincost}_E(c_j, c_i).$$

Thus we can easily extract the cost of microbribery M_f from the cost of flow f.

Our algorithm crucially depends on this correspondence between flows and microbriberies. (In the proofs of Lemmas 6.11 and 6.13 that cover the case of an even number of voters we simply show how to modify the instances $I(T)$ to handle ties, and we show correspondences between the new networks and microbriberies; the rest of these proofs is the same as here.)

Note that for small values of T no flow of value F exists for $I(T)$. The reason for this is that the edges coming into the sink t might not have enough capacity to hold a

[4]Note that for each $c_i \in C$ any flow of value F within $I(T)$ needs to send exactly $\mathit{score}_E^\alpha(c_i)$ units from s to c_i.

```
procedure Copeland$^\alpha$-odd-microbribery($E = (C, V), k, p$)
begin
    if $p$ is a winner of $E$ then accept;
    $F = \sum_{c_i \in C} score^\alpha_E(c_i) = \frac{\|C\|(\|C\|-1)}{2}$;
    for $T = 0$ to $\|C\| - 1$ do
    begin
        build an instance $I(T)$ of min-cost-flow;
        if $I(T)$ has no flow of value $F$ then
            restart the for loop with the next value of $T$;
        $f$ = a minimum-cost flow for $I(T)$;
        if $f(c_0, t) < T$ then restart the loop;
        $\kappa = \mathit{flowcost}(f) - B \cdot (F - T)$;
        if $\kappa \leq k$ then accept;
    end;
    reject;
end
```

Figure 6.2: The microbribery algorithm for Copeland$^\alpha$ elections with an odd number of voters.

flow of value F. In such situations it is impossible to guarantee that every candidate gets at most T points; there are too many Copeland$^\alpha$ points to distribute.

Figure 6.2 gives our algorithm for constructive microbribery in Copeland$^\alpha$. There is a polynomial-time algorithm for the min-cost-flow problem and thus this algorithm can be implemented to run in polynomial time.

Let us now prove that this algorithm is correct. We have presented above how a flow f of value F within $I(T)$ (with $0 \leq T \leq F$) defines a microbribery. Based on this, it is clear that if our algorithm accepts then there is a microbribery of cost at most k that ensures p's victory.

On the other hand, suppose now that there exists a microbribery of cost at most k that ensures p's victory in the election. We will show that our algorithm accepts in this case.

Let M be a minimum-cost microbribery (of cost at most k) that ensures p's victory. As pointed out above, M can be represented as a collection of pairs (c_i, c_j) of distinct candidates for whom we flip the result of the head-to-head contest from c_i winning to c_j winning. The cost of M is

$$\sum_{(c_i,c_j)\in M} \mathit{wincost}_E(c_j, c_i).$$

Since applying microbribery M ensures that p is a winner, we have that each candidate among $c_1, c_2, \ldots, c_m$ has at most as many Copeland$^\alpha$ points as p does. Let E' be the election that results from E after applying microbribery M to E (i.e., after flipping the results of the contests specified by M in an optimal way, as given by $\mathit{wincost}_E$). Let T' be $\mathit{score}^\alpha_{E'}(p)$, p's Copeland$^\alpha$ score after implementing M. Clearly, $0 \leq T' \leq \|C\| - 1$.

Consider instance $I(T')$ and let f_M be the flow that corresponds to the microbribery M. In this flow each edge of the form (s, c_i) carries flow of its maximum capacity, $\mathit{score}^\alpha_E(c_i)$, each edge of the form (c_i, c_j) carries one unit of flow exactly if e is listed in M and carries zero units of flow otherwise, and each edge of the form (c_i, t) carries $\mathit{score}^\alpha_{E'}(c_i)$ units of flow. It is easy to see that this is a legal flow. The cost of f_M is

$$\mathit{flowcost}(f_M) = B \cdot (F - T') + \sum_{(c_i,c_j)\in M} \mathit{wincost}_E(c_j, c_i).$$

After applying M, p gets T' Copeland$^\alpha$ points that travel to the sink t via edge (c_0, t) with cost $a(c_0, t) = 0$, and all the remaining $F - T'$ points travel via edges (c_i, t), $i \in \{1, 2, \ldots, m\}$, with cost $a(c_i, t) = B$. The remaining part of $\mathit{flowcost}(f_M)$ is the cost of the units of flow traveling through the edges (c_i, c_j) that directly correspond to the cost of microbribery M.

Now consider some minimum-cost flow $f_{\min}$ for $I(T')$. Since f_M exists, a minimum-cost flow must exist as well. Clearly, we have

$$\mathit{flowcost}(f_{\min}) \leq \mathit{flowcost}(f_M).$$

Let T'' be the number of units of flow that $f_{\min}$ assigns to travel over the edge (c_0, t), i.e., $T'' = f_{\min}(c_0, t)$. The only edges with nonzero cost for sending flow through them are those in the set $\{(c_i, c_j) \mid c_i, c_j \in C \wedge \mathrm{vs}_E(c_i, c_j) > 0\} \cup \{(c_i, t) \mid i \in \{1, \ldots, m\}\}$ and thus the cost of $f_{\min}$ can be expressed as (recall that $\mathrm{vs}_E(c_i, c_j) > 0$ implies $i \neq j$)

$$\mathit{flowcost}(f_{\min}) = B \cdot (F - T'') + \sum_{c_i, c_j \in C \wedge \mathrm{vs}_E(c_i, c_j) > 0} f_{\min}(c_i, c_j) \cdot \mathit{wincost}_E(c_j, c_i).$$

$B > \sum_{i,j,i \neq j} \mathit{wincost}_E(c_i, c_j)$, for each $c_i, c_j \in C$ such that $\mathrm{vs}_E(c_i, c_j) > 0$ we have $f_{\min}(c_i, c_j) \in \{0, 1\}$, and $\mathit{flowcost}(f_{\min}) \leq \mathit{flowcost}(f_M)$, so it must hold that $T'' = T'$ and

$$\sum_{c_i, c_j \in C \wedge \mathrm{vs}_E(c_i, c_j) > 0} f_{\min}(c_i, c_j) \cdot \mathit{wincost}_E(c_j, c_i) \leq \sum_{(c_i, c_j) \in M} \mathit{wincost}_E(c_j, c_i).$$

Thus flow $f_{\min}$ corresponds to a microbribery that guarantees p's victory and has cost at most as high as that of M. Since M was chosen to have minimum cost among all such microbriberies, flow $f_{\min}$ corresponds to a microbribery of minimum cost and our algorithm correctly accepts within the for loop of Figure 6.2, at the very latest when in the for loop T is set to T'. ❑

We now turn to the algorithms showing that Copeland^0 and Copeland^1, with irrational voters allowed, are vulnerable to microbribery when the number of voters is even. In this case we need to take into account that it is sometimes more desirable to have candidates tie each other in a head-to-head contest than to have one of them win the contest.

Lemma 6.10. *Let $E = (C, V)$ be an election with candidate set $C = \{c_0, c_1, \ldots, c_m\}$ and with an even number of voters, specified via preference tables over C. If the election is conducted using* Copeland^0 *then no minimum-cost microbribery that ensures victory for c_0 involves either (a) flipping a result of a head-to-head contest between any two distinct candidates $c_i, c_j \in C - \{c_0\}$ from c_i winning to c_j winning, or (b) changing a result of a head-to-head contest between any two distinct candidates in $C - \{c_0\}$ from a tie to one of them winning.*

Proof. Our proof follows by way of contradiction. Let $E = (C, V)$ be an election as specified in the lemma. For the sake of a contradiction suppose there is a minimum-cost microbribery M that makes c_0 win and that there are two distinct candidates, c_i and c_j, in $C - \{c_0\}$ such that microbribery M involves switching the result of the head-to-head contest between these candidates from c_i winning to c_j winning or from a tie to one of them winning. Consider the microbribery M' that is identical to M, except that it makes c_i tie with c_j in a head-to-head contest, either via an appropriate number of microbribes if c_i and c_j do not tie originally, or via leaving the corresponding preference-table entries untouched if they do tie initially. Clearly, this microbribery M' has a lower cost than M and it still ensures c_0's victory. This is a contradiction. ❑

With Lemma 6.10 at hand, we can show that microbribery is easy for Copeland^0 for the case of an even number of voters.

Lemma 6.11. *There is a polynomial-time algorithm that solves the microbribery problem for* Copeland^0 *elections with an even number of voters.*

Proof. Our input is election $E = (C, V)$, where $C = \{c_0, c_1, \dots, c_m\}$, $p = c_0$, and V is a collection of an even number of voters, each specified via a preference table over C. Our algorithm is essentially the same as that used in the proof of Lemma 6.9, except that instead of using instances $I(T)$ we now use instances $J(T)$ defined below. In this proof we show how to construct these instances and how they correspond to microbriberies within E. The proof of Lemma 6.9 shows how to use such a correspondence to solve the microbribery problem at hand.

Let T be a nonnegative integer, $0 \leq T \leq \|C\| - 1$. Instance $J(T)$ is somewhat different from the instance $I(T)$ used in the proof of Lemma 6.9. In particular, due to Lemma 6.10, we model only microbriberies that have the following effects on our election:

1. For any two distinct candidates c_i, c_j in $C - \{c_0\}$, the result of the head-to-head contest between c_i and c_j may possibly turn into a tie.

2. For each candidate c_i in $C - \{c_0\}$, the result of a head-to-head contest between c_0 and c_i may possibly turn into either a tie (from c_i defeating c_0) or into c_0 defeating c_i (from either a tie or from c_i defeating c_0).

Our instance $J(T)$ contains special nodes, namely the elements of the sets C' and C'' below, to handle these possible interactions. We define

$$\begin{aligned} C' &= \{c_{ij} \mid i, j \in \{1, 2, \ldots, m\} \wedge \mathrm{vs}_E(c_i, c_j) > 0\} \text{ and} \\ C'' &= \{c_{i0} \mid i \in \{1, 2, \ldots, m\} \wedge \mathrm{vs}_E(c_i, c_0) \geq 0\}. \end{aligned}$$

For each possible threshold T, define $J(T)$ to be the flow network with node set $K = C \cup C' \cup C'' \cup \{s, t\}$, where s is the source, t is the sink, and the edge capacities and costs are as stated in Figure 6.3. (As before, we set B to be a number that is greater than the cost of any possible microbribery within E, e.g., $B = \|V\| \cdot \|C\|^2 + 1$.) The target flow value is

$$F = \sum_{v \in K} c(s, v).$$

Instance $J(T)$ is fairly complicated but it in fact does closely follow the instance of microbribery that we have at hand. As in the proof of Lemma 6.9, the units of flow that travel through the network are interpreted as Copeland^0 points, and flows are interpreted as specifying microbriberies. We now describe a bit more precisely how we interpret our flow network $J(T)$. In particular, we will argue that each flow f of value F that travels through the network $J(T)$ corresponds to a microbribery within E that gives each candidate $c_i \in C$ exactly $f(c_i, t)$ Copeland^0 points:

1. For each $c_i \in C$, the units of flow that enter c_i from s correspond to the number of c_i's Copeland^0 points in E, prior to any microbribery.

2. For each $c_i \in C$, the units of flow that go directly from c_i to t correspond to the number of c_i's Copeland^0 points after a microbribery as specified by the flow.

3. For each pair of distinct candidates $c_i, c_j \in C - \{c_0\}$ such that c_i defeats c_j in their head-to-head contest in E, we have an edge $e = (c_i, c_{ij})$ with capacity one and cost $\mathit{tiecost}_E(c_j, c_i)$. A unit of flow that travels through e corresponds to a

Edge	Parameters
$e = (s, c_i)$, where $c_i \in C$	$c(e) = \mathit{score}^0_E(c_i)$ $a(e) = 0$
$e = (c_i, c_{ij})$, where $c_i, c_j \in C - \{c_0\}$ and $\mathrm{vs}_E(c_i, c_j) > 0$	$c(e) = 1$ $a(e) = \mathit{tiecost}_E(c_j, c_i)$
$e = (c_i, t)$, where $c_i \in C - \{c_0\}$	$c(e) = T$ $a(e) = B$
$e = (c_{ij}, t)$, where $c_i, c_j \in C - \{c_0\}$ and $\mathrm{vs}_E(c_i, c_j) > 0$	$c(e) = 1$ $a(e) = B$
$e = (c_i, c_{i0})$, where $c_i \in C - \{c_0\}$ and $\mathrm{vs}_E(c_i, c_0) > 0$	$c(e) = 1$ $a(e) = \mathit{tiecost}_E(c_0, c_i)$
$e = (c_{i0}, c_0)$, where $c_i \in C - \{c_0\}$ and $\mathrm{vs}_E(c_i, c_0) \geq 0$	$c(e) = 1$ $a(e) = \mathit{wincost}_E(c_0, c_i) - \mathit{tiecost}_E(c_0, c_i)$
$e = (s, c_{i0})$, where $c_i \in C - \{c_0\}$ and $\mathrm{vs}_E(c_i, c_0) = 0$	$c(e) = 1$ $a(e) = 0$
$e = (c_{i0}, t)$, where $c_i \in C - \{c_0\}$ and $\mathrm{vs}_E(c_i, c_0) \geq 0$	$c(e) = 1$ $a(e) = B$
$e = (c_0, t)$	$c(e) = T$ $a(e) = 0$
every other edge e	$c(e) = 0$ $a(e) = 0$

Figure 6.3: Edge capacities and costs for min-cost-flow instance $J(T)$, built from election E.

microbribe that makes c_i tie with c_j: c_i loses the point, we pay $\mathit{tiecost}_E(c_j, c_i)$, and then the unit of flow goes directly to t. (From Lemma 6.10 we know that we do not need to handle any other possible interactions between c_i and c_j in their head-to-head contest.)

4. For each candidate $c_i \in C - \{c_0\}$ such that c_i defeats c_0 in a head-to-head contest,

we need to allow for the possibility that a microbribery causes c_0 to either tie with c_i or to defeat c_i. Fix an arbitrary such c_i. A unit of flow that travels directly from c_i to c_{i0} and then directly to t corresponds to a microbribery after which c_0 ties with c_i: c_i loses the point but c_0 does not receive it and the cost of the flow increases by $\mathit{tiecost}_E(c_0, c_i)$.

On the other hand, if that unit of flow travels from c_i to c_{i0} and then directly to c_0, then this corresponds to a microbribery after which c_0 defeats c_i. The point travels from c_i to c_0 and the cost of the flow increases by $\mathit{wincost}_E(c_0, c_i)$.

If there is no flow entering node c_{i0} then this means that our microbribery does not change the result of a head-to-head contest between c_0 and c_i.

5. For each candidate $c_i \in C - \{c_0\}$ such that c_i ties with c_0 in a head-to-head contest before any microbribery is attempted, we need to allow for c_0 defeating c_i after the microbribery. Fix any such c_i. A unit of flow that travels directly from s to c_{i0} and then directly to c_0 corresponds to a microbribery after which c_0 defeats c_i in a head-to-head contest: c_0 gets an additional point and the cost of the flow increases by $\mathit{wincost}_E(c_0, c_i) - \mathit{tiecost}_E(c_0, c_i) = \mathit{wincost}_E(c_0, c_i)$.

   On the other hand, a unit of flow that travels from s directly to c_{i0} and then directly to t corresponds to a microbribery that does not change the result of a head-to-head contest between c_0 and c_i.

Based on these comments, we can see the natural correspondence between flows in $J(T)$ and microbriberies. In particular, each flow f of value F that travels through the network $J(T)$ corresponds to a microbribery within E that gives each candidate $c_i \in C$ exactly $f(c_i, t)$ Copeland^0 points.

Now let M_f be a microbribery defined by flow f of value F within $J(T)$, $0 \leq T \leq \|C\| - 1$, assuming that one exists. Let $\mathit{cost}(M_f)$ be the minimum cost of implementing microbribery M_f. A close inspection of instance $J(T)$ shows that the cost of such a flow f is

$$\mathit{flowcost}(f) = B \cdot (F - f(c_0, t)) + \mathit{cost}(M_f).$$

Because of the above equation, the fact that all units of flow that are not accounted for as c_0's points (i.e., the units of flow that do not travel through the edge (c_0, t)) impose cost B on the flow, and via arguments analogous to those in Lemma 6.9, the following holds: If for a given $J(T)$, $0 \leq T \leq \|C\| - 1$, there exists a flow of value F then a minimum-cost flow $f_{\min}$ of value F corresponds to a minimum-cost microbribery that ensures that c_0 receives T points and each other candidate receives at most T points. (Intuitively, the reason for this is that B is so large that in a minimum-cost flow one would always send as few units of flow through edges of cost B as possible) Thus, to solve the constructive microbribery problem for Copeland^0 elections with an even number of voters it is enough to run the algorithm from Figure 6.2, using the instances $J(T)$ instead of $I(T)$ and using the new value of F. ❑

We now show that Copeland^1, with irrational voters allowed, is vulnerable to microbribery when there are an even number of voters. The following lemma reduces the set of microbriberies we need to model in this case.

Lemma 6.12. *Let $E = (C, V)$ be an election with candidate set $C = \{c_0, c_1, \ldots, c_m\}$ and with an even number of voters, specified via preference tables over C. If the election is conducted using* Copeland^1 *then no minimum-cost microbribery that ensures victory for c_0 involves obtaining a tie in a head-to-head contest between any two distinct candidates in $C - \{c_0\}$.*

Proof. Our proof is again by way of contradiction. Let $E = (C, V)$ be an election as specified in the lemma. Suppose there is a minimum-cost microbribery that ensures c_0's victory and that involves obtaining a tie in a head-to-head contest between two distinct candidates in $C - \{c_0\}$, say c_i and c_j. That is, before this microbribery we have that either c_i defeats c_j or c_j defeats c_i in their head-to-head contest but afterward they are tied. Clearly, a microbribery that is identical to this one except that does not change the result of the head-to-head contest between c_i and c_j (i.e., one that does not microbribe any voters to flip their preference-table entries regarding c_i versus c_j) has a smaller cost and still ensures c_0's victory. This is a contradiction. ❑

Lemma 6.13. *There is a polynomial-time algorithm that solves microbribery problem for* Copeland^1 *elections with an even number of voters.*

Proof. We give a polynomial-time algorithm for constructive microbribery in Copeland^1 elections with an even number of voters. Our input is a budget $k \in \mathbb{N}$ and an election $E = (C, V)$, where $C = \{c_0, c_1, \ldots, c_m\}$, $p = c_0$, and V contains an even number of voters specified via their preference tables over C. Our goal is to ensure p's victory via at most k microbribes.

We use essentially the algorithm from the proof of Lemma 6.9, except that instead of using instances $I(T)$ we now employ instances $L(T)$ that are designed to handle tie issues as appropriate for Copeland^1. Lemma 6.12 tells us that our min-cost-flow instances $L(T)$ do not need to model microbriberies that incur ties between pairs of distinct candidates in $C - \{c_0\}$. We also don't need to model microbriberies that change the outcome of the head-to-head contest between c_0 and any candidate in $C - \{c_0\}$ from c_0 winning to c_0 not winning or from a tie to c_0 losing. However, we do need to model possible microbribery-induced ties between c_0 and each c_i in $C - \{c_0\}$.

Define B to be a number that is greater than the cost of any possible microbribery within E (e.g., $B = \|V\| \cdot \|C\|^2 + 1$ will do). Further, define the following three sets of nodes:

$$\begin{aligned} C' &= \{c'_i \mid c_i \in C\}, \\ C'' &= \{c_{ij} \mid i < j \wedge c_i, c_j \in C \wedge \mathrm{vs}_E(c_i, c_j) = 0\}, \text{ and} \\ C''' &= \{c_{0i} \mid c_i \in C \wedge \mathrm{vs}_E(c_i, c_0) > 0\}. \end{aligned}$$

Our flow network $L(T)$ has the node set $K = C \cup C' \cup C'' \cup C''' \cup \{s, t\}$, where s is the source, t is the sink, and the capacities and costs of edges are defined in Figure 6.4. Each instance $L(T)$ asks for a minimum-cost flow of value

$$F = \sum_{c_i \in C} c(s, c_i).$$

Note that a flow f of value F within $L(T)$, $0 \leq T \leq \|C\| - 1$, corresponds to a microbribery M_f within election E that leaves each candidate c_i with exactly $f(c_i, c'_i)$

Edge	Parameters
$e = (s, c_i)$, where $c_i \in C$	$c(e) = \mathit{score}^{\mathrm{L}}_{E}(c_i)$ $a(e) = 0$
$e = (c_i, c_j)$, where $c_i, c_j \in C - \{c_0\}$ and $\mathrm{vs}_E(c_i, c_j) > 0$	$c(e) = 1$ $a(e) = \mathit{wincost}_E(c_j, c_i)$
$e = (c_i, c_{ij})$, where $i < j$, $c_i, c_j \in C$ and $\mathrm{vs}_E(c_i, c_j) = 0$	$c(e) = 1$ $a(e) = \mathit{wincost}_E(c_j, c_i)$
$e = (c_j, c_{ij})$, where $i < j$, $c_i, c_j \in C$ and $\mathrm{vs}_E(c_i, c_j) = 0$	$c(e) = 1$ $a(e) = \mathit{wincost}_E(c_i, c_j)$
$e = (c_{ij}, t)$, where $i < j$, $c_i, c_j \in C$ and $\mathrm{vs}_E(c_i, c_j) = 0$	$c(e) = 1$ $a(e) = B$
$e = (c_i, c_{0i})$, where $c_i \in C - \{c_0\}$ and $\mathrm{vs}_E(c_i, c_0) > 0$	$c(e) = 1$ $a(e) = \mathit{wincost}(c_0, c_i)$
$e = (c'_i, c_{0i})$, where $c_i \in C - \{c_0\}$ and $\mathrm{vs}_E(c_i, c_0) > 0$	$c(e) = 1$ $a(e) = \mathit{tiecost}(c_0, c_i)$
$e = (c_{0i}, c_0)$, where $c_i \in C - \{c_0\}$ and $\mathrm{vs}_E(c_i, c_0) > 0$	$c(e) = 1$ $a(e) = 0$
$e = (c_i, c'_i)$, where $c_i \in C$	$c(e) = T$ $a(e) = 0$
$e = (c'_i, t)$, where $c_i \in C - \{c_0\}$	$c(e) = T$ $a(e) = B$
$e = (c'_0, t)$	$c(e) = T$ $a(e) = 0$
every other edge e	$c(e) = 0$ $a(e) = 0$

Figure 6.4: Edge capacities and costs for min-cost-flow instance $L(T)$, built from election E.

Copeland^1 points. As in the proofs of Lemmas 6.9 and 6.11, points traveling through the network $L(T)$ are here interpreted as Copeland^1 points and flows are interpreted as specifying microbriberies. More specifically, we interpret the units of flow traveling through $L(T)$ as follows:

1. For each $c_i \in C$, the units of flow that enter c_i from s are interpreted as the Copeland^1 points that c_i has before any microbribery is attempted.

2. For each $c_i \in C$, the units of flow that travel directly from c_i to c_i' are interpreted as the Copeland^1 points that c_i has after the microbribery defined by f has been performed.

3. If a candidate $c_i \in C-\{c_0\}$ originally defeats c_0 but our flow models a microbribery in which c_0 and c_i end up tied in their head-to-head contest, then we have a single Copeland^1 point that travels from c_i to c_i', then to c_0 through c_{0i}, at cost $\mathit{tiecost}_E(c_0, c_i)$, and then to c_0'. This way the same unit of flow is accounted both for the score of c_0 and for the score of c_i. Note that such a unit of flow then travels to t through edge (c_0, t) at zero cost.

4. For any two distinct candidates c_i and c_j such that $c_i, c_j \in C-\{c_0\}$ where c_i defeats c_j in a head-to-head contest in E, a unit of flow traveling from c_i to c_j corresponds to a microbribery that flips the result of their head-to-head contest. Thus c_j receives the Copeland^1 point and the cost of the flow increases by $\mathit{wincost}_E(c_j, c_i)$.

5. For any $c_i \in C - \{c_0\}$ where c_i defeats c_0 in a head-to-head contest in E, a unit of flow traveling from c_i to c_0 via c_{0i} corresponds to a microbribery that flips the result of their head-to-head contest. Thus c_i receives the Copeland^1 point and the cost of the flow increases by $\mathit{wincost}_E(c_0, c_i)$. Note that since the edge (c_{0i}, c_0) has capacity one we ensure that at most one unit of flow travels from c_i to c_0 (either on a path c_i, c_{0i}, c_0 (modeling a microbribery that flips the result of the head-to-head contest between c_i and c_0 to c_0 winning) or on a path c_i, c_i', c_{0i}, c_0 (modeling a microbribery that enforces a tie between c_0 and c_i)).

6. For each $c_i, c_j \in C$, we have to take into account the possibility that c_i and c_j are tied in their head-to-head contest within E, but via microbribery we want to change the result of this contest. Let $c_i, c_j \in C$ be two such candidates and let $i < j$. Here, a unit of flow traveling from c_i to c_{ij} (or, analogously, from c_j to c_{ij}) is interpreted as a microbribery that ensures c_j's (c_i's) victory in the head-to-head contest. Since c_i and c_j were already tied, c_j (c_i) already has his or her point for the victory and c_i (c_j) gets rid of his or her point through the node c_{ij}. The cost of the flow increases by $\mathit{wincost}_E(c_j, c_i)$ (respectively, by $\mathit{wincost}_E(c_i, c_j)$). Also, we point out that via the introduction of node c_{ij} we ensure that only one of c_i and c_j, let us call him or her c_k, can lose a point by sending it from c_k to c_{ij} and then to t; the capacity of edge (c_{ij}, t) is only one.

Through the above description, we can see the natural correspondence between flows in $L(T)$ and microbriberies. In particular, each flow f of value F corresponds to a microbribery M_f within E that gives each candidate c_i exactly $f(c_i, c_i')$ points.

Let us now analyze the cost of f. It is easy to see that each unit of flow that is not accounted for as a Copeland^1 point of c_0 reaches the sink t via an edge of cost B. Also, the only other edges through which units of flow travel at nonzero cost are those that define the microbribery M_f. Thus the cost of our flow f can be expressed as

$$\mathit{flowcost}(f) = B \cdot (F - f(c_0, c_0')) + \mathit{cost}(M_f).$$

Fix T such that $0 \leq T \leq \|C\| - 1$. Given the above properties of $L(T)$ and by the arguments presented in the proof of Lemma 6.9, if a flow of value F exists within the flow network of instance $L(T)$, then each minimum-cost flow in $L(T)$ corresponds to a microbribery that ensures that c_0 has exactly T Copeland^1 points and every other candidate has at most T Copeland^1 points. Thus if there exists a value T' such that

1. there is a flow of value F in $L(T')$ and

2. the cost of a minimum-cost flow f of value F in $L(T')$ is K,

then there is a microbribery of cost $K - B \cdot (F - T')$ that ensures c_0's victory.

On the other hand, via Lemma 6.12 and our correspondence between flows for $L(T)$ and microbriberies in E, there is a value T'' such that a minimum-cost flow in $L(T'')$ corresponds to a minimum-cost microbribery that ensures p's victory. Thus the algorithm from Figure 6.2, used with the instances $L(T)$ instead of $I(T)$ and with our new value of F, solves in polynomial time the constructive microbribery problem for Copeland^1 with an even number of voters. ❑

Together, Lemmas 6.9, 6.11, and 6.13 show that both Copeland^0 and Copeland^1 are vulnerable to microbribery. It is interesting to note that all of our microbribery proofs above would work just as well if we considered a slight twist to the definition of the microbribery problem, namely–if instead of saying that each flip in a voter's preference table has unit cost we would allow each voter to have a possibly different price for flipping each separate entry in his or her preference table. This change would only affect computing the values of the functions *wincost* and *tiecost*. (Technically, we would also have to modify Lemmas 6.10 and 6.12, which in our unit-cost setting say that an optimal microbribery *never* involves certain specified pairs of candidates, whereas in the priced setting we would need to rephrase them to state that there *exist* optimal microbriberies that do not involve those specified pairs of candidates.)

An interesting direction for further study of the complexity of bribery within Copeland$^\alpha$ systems is to consider a version of the microbribery problem for the case of rational voters. There, a briber would pay unit cost for a switch of two adjacent candidates on a given voter's preference list.

For Copeland$^\alpha$, we would also like to know the complexity of microbribery when α is a rational number strictly between 0 and 1. Our network-flow-based approach does not seem to generalize easily to values of α strictly between 0 and 1 (when the number of voters is even) because in a flow network it is hard to "split" a unit of flow in a tie. A promising approach would be to have several units of flow model one Copeland$^\alpha$ point (e.g., for the case of $\alpha = \frac{1}{2}$ we could try to use two units of flow to model a single Copeland$^{0.5}$ point), but then it seems very difficult (if not impossible) to find edge costs that appropriately model the microbribery. (It is possible to do so in a very restricted setting, namely where $\alpha = \frac{1}{2}$ and there are exactly two voters that can be bribed.) Our results on manipulation suggest that microbribery for α strictly between 0 and 1 might be NP-hard, but we haven't found a way to translate our manipulation proof to the world of microbribery.

On a related note, Kern and Paulusma (KP01) have shown that the following problem, which they call $\mathrm{SC}(0, \alpha, 1)$, is NP-complete: Let α be a rational number such that $0 < \alpha < 1$ and $\alpha \neq \frac{1}{2}$. We are given an undirected graph $G = (V(G), E(G))$, where each vertex $u \in V(G)$ is assigned a rational value c_u of the form $i + j\alpha$, for nonnegative integers i and j. The question, which we have rephrased to state in terms of (a variant of) our notion of Copeland$^\alpha$, is whether it is possible to (possibly partially) orient the edges of G such that for each vertex $u \in V(G)$ it holds that u's Copeland$^\alpha$ score is at most c_u. Here, by "Copeland$^\alpha$ score of a vertex u" we mean, as is natural, the number of vertices u "defeats" (i.e., the number of vertices v such that there is a directed edge from u to v) plus α times the number of vertices that u "ties" (i.e., the number of vertices such that there is an undirected edge between u and v).

Problem $\mathrm{SC}(0, \alpha, 1)$ is very closely related to our microbribery problem. However we do not see an immediate reduction from $\mathrm{SC}(0, \alpha, 1)$ to microbribery. A natural approach would be to embed graph G into an election (e.g., using a lemma similar to Lemma 6.6) in such a way that our preferred candidate p can become a winner (via a microbribery) if and only if it is possible to orient the edges of G in a way respecting the constraints defined by the values c_u (for each u in $V(G)$). We would, of course, have to set the budget of our microbribery to allow modifying each of the edges in G and none of the edges outside of G. However, this is difficult. The proof of Kern and Paulusma uses values c_u that can be implemented only via using tied head-to-head contests. The agent performing microbribery could, potentially, affect those head-to-head contests, thus spoiling our reduction.

6.4 Conclusions and Research Directions

In this chapter we have shown that Copeland$^\alpha$, in the rational-voter model, is resistant to essentially all standard types of bribery and manipulation (though, in the case of manipulation our results are limited to rational α's where $0 < \alpha < \frac{1}{2}$; (FHS08) shows an analogous result for rational α's such that $\frac{1}{2} < \alpha < 1$, but the cases where $\alpha \in \{0, \frac{1}{2}, 1\}$ remain open). It is also known that Copeland$^\alpha$ is resistant to essentially all standard types of (constructive) control (FHHR07; FHHR08). Thus, from the point of view of computational social choice, Copeland$^\alpha$ is one of the most promising election systems.

In Section 6.3 we have considered microbribery in the irrational-voter model and we have shown that Copeland^0 and Copeland^1 are both vulnerable to microbribery. A natural open question is to consider microbribery for the irrational-voter for the remaining values of α, that is for rational alpha such that $0 < \alpha < 1$. We conjecture that the problem for these values of α is NP-complete. At the end of Section 6.3 we have outlined some possible attacks to obtain the proof of this conjecture and we have explained the difficulties one would have to overcome in order to make these attacks successful.

Another interesting research direction is to seek an election system that is resistant to all standard attacks (bribery, manipulation, control) in both the constructive setting and the destructive setting (see (HHR07b) regarding control). In this thesis we have not considered the destructive setting, but we mention that for the case of control, Copeland$^\alpha$ is vulnerable to some such attacks (FHHR07; FHHR08).

Bibliography

[AB00] D. Austen-Smith and J. Banks. *Positive Political Theory I: Collective Preference.* University of Michigan Press, 2000.

[ACG$^+$99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and their Approximability Properties.* Springer-Verlag, 1999.

[AMO93] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, 1993.

[Arr63] K. Arrow. *Social Choice and Individual Values.* John Wiley and Sons, 1951 (revised editon, 1963).

[BBD$^+$04] C. Boutilier, R. Brafman, C. Domshlak, H. Hoos, and D. Poole. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21:135–191, 2004.

[BC93] D. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity.* Prentice-Hall, 1993.

[BDG90] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity II.* EATCS Monographs in Theoretical Computer Science. Springer-Verlag, 1990.

[BDG95] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I.* EATCS Texts in Theoretical Computer Science. Springer-Verlag, 2nd edition, 1995.

[BFH+08] E. Brelsford, P. Faliszewski, E. Hemaspaandra, H. Schnoor, and I. Schnoor. Approximability of manipulating elections. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pages 44–49. AAAI Press, July 2008.

[BO91] J. Bartholdi, III and J. Orlin. Single transferable vote resists strategic voting. *Social Choice and Welfare*, 8(4):341–354, 1991.

[Bre07] E. Brelsford. Approximation and elections. Master's thesis, Rochester Institute of Technology, Rochester, NY, May 2007.

[BS06] S. Brams and R. Sanver. Critical strategies under approval voting: Who gets ruled in and ruled out. *Electoral Studies*, 25(2):287–305, 2006.

[BTT89a] J. Bartholdi, III, C. Tovey, and M. Trick. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6(3):227–241, 1989.

[BTT89b] J. Bartholdi, III, C. Tovey, and M. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165, 1989.

[BTT92] J. Bartholdi, III, C. Tovey, and M. Trick. How hard is it to control an election? *Mathematical and Computer Modeling*, 16(8/9):27–40, 1992.

[CELM07] Y. Chevaleyre, U. Endriss, J. Lang, and N. Maudet. A short introduction to computational social choice. In *Proceedings of the 33rd International Conference on Current Trends in Theory and Practice of Computer Science*, pages 51–69. Springer-Verlag, January 2007.

[CFRS07] R. Christian, M. Fellows, F. Rosamond, and A. Slinko. On complexity of lobbying in multiple referenda. *Review of Economic Design*, 11(3):217–224, 2007.

[CLRS01] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press/McGraw Hill, second edition, 2001.

[Con07] V. Conitzer. Eliciting single-peaked preferences using comparison queries. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 408–415. ACM Press, May 2007.

[CS02] V. Conitzer and T. Sandholm. Vote elicitation: Complexity and strategy-proofness. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pages 392–397. AAAI Press, July/August 2002.

[CS03] V. Conitzer and T. Sandholm. Universal voting protocol tweaks to make manipulation hard. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 781–788. Morgan Kaufmann, August 2003.

[CS06] V. Conitzer and T. Sandholm. Nonexistence of voting rules that are usually hard to manipulate. In *Proceedings of the 21st National Conference on Artificial Intelligence*, pages 627–634. AAAI Press, July 2006.

[CSL07] V. Conitzer, T. Sandholm, and J. Lang. When are elections with few candidates hard to manipulate? *Journal of the ACM*, 54(3):Article 14, 2007.

[DKNS01] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of the 10th International World Wide Web Conference*, pages 613–622. ACM Press, March 2001.

[DS00] J. Duggan and T. Schwartz. Strategic manipulability without resoluteness or shared beliefs: Gibbard–Satterthwaite generalized. *Social Choice and Welfare*, 17(1):85–93, 2000.

[EHRS07] G. Erdélyi, L. Hemaspaandra, J. Rothe, and H. Spakowski. On approximating optimal weighted lobbying, and frequency of correctness versus average-case polynomial time. In *Proceedings of the 16th International Symposium on Fundamentals of Computation Theory*, pages 300–311. Springer-Verlag *Lecture Notes in Computer Science #4639*, August 2007.

[EL05a] E. Elkind and H. Lipmaa. Hybrid voting protocols and hardness of manipulation. In *The 16th Annual International Symposium on Algorithms and Computation, ISAAC 2005*, pages 206–215. Springer-Verlag *Lecture Notes in Computer Science #3872*, December 2005.

[EL05b] E. Elkind and H. Lipmaa. Small coalitions cannot manipulate voting. In *Proceedings of the 9th International Conference on Financial Cryptography and Data Security*, pages 285–297. Springer-Verlag *Lecture Notes in Computer Science #3570*, February/March 2005.

[ENR08] G. Erdélyi, M. Nowak, and J. Rothe. Sincere-strategy preference-based approval voting broadly resists control. In *Proceedings of the 33rd International Symposium on Mathematical Foundations of Computer Science*, pages 311–322. Springer-Verlag *Lecture Notes in Computer Science #5162*, August 2008.

[ER97] E. Ephrati and J. Rosenschein. A heuristic technique for multi-agent planning. *Annals of Mathematics and Artificial Intelligence*, 20(1–4):13–67, 1997.

[Fal08] P. Faliszewski. Nonuniform bribery (short paper). In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems*, pages 1569–1572, May 2008.

[FHH06] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. The complexity of bribery in elections. In *Proceedings of the 21st National Conference on Artificial Intelligence*, pages 641–646. AAAI Press, July 2006.

[FHHR] P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. A richer understanding of the complexity of election systems. In S. Ravi and S. Shukla, editors, *Fundamental Problems in Computing: Essays in Honor of Professor Daniel J. Rosenkrantz*. Springer. To appear. Preliminary version available as (FHHR06).

[FHHR06] P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. A richer understanding of the complexity of election systems. Technical Report TR-903, Department of Computer Science, University of Rochester, Rochester, NY, September 2006.

[FHHR07] P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Llull and Copeland voting broadly resist bribery and control. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, pages 724–730. AAAI Press, July 2007.

[FHHR08] P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Copeland voting fully resists constructive control. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management*, pages 165–176. Springer-Verlag *Lecture Notes in Computer Science #5034*, June 2008.

[FHS08] P. Faliszewski, E. Hemaspaandra, and H. Schnoor. Copeland voting: Ties matter. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems*, pages 983–990, May 2008.

[Gib73] A. Gibbard. Manipulation of voting schemes. *Econometrica*, 41(4):587–601, 1973.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[GMHS99] S. Ghosh, M. Mundhe, K. Hernandez, and S. Sen. Voting for movies: The anatomy of recommender systems. In *Proceedings of the 3rd Annual Conference on Autonomous Agents*, pages 434–435. ACM Press, 1999.

[Gre08] E. Green, September 2008. Personal communication.

[HH] C. Homan and L. Hemaspaandra. Guarantees for the success frequency of an algorithm for finding Dodgson-election winners. *Journal of Heuristics.* To appear. Full version available as (HH05).

[HH05] C. Homan and L. Hemaspaandra. Guarantees for the success frequency of an algorithm for finding Dodgson-election winners. Technical Report TR-881, Department of Computer Science, University of Rochester, Rochester, NY, September 2005. Revised, June 2007.

[HH07] E. Hemaspaandra and L. Hemaspaandra. Dichotomy for voting systems. *Journal of Computer and System Sciences*, 73(1):73–83, 2007.

[HHR97] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Exact analysis of Dodgson elections: Lewis Carroll's 1876 voting system is complete for parallel access to NP. *Journal of the ACM*, 44(6):806–825, 1997.

[HHR07a] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Anyone but him: The complexity of precluding an alternative. *Artificial Intelligence*, 171(5-6):255–285, April 2007.

[HHR07b] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Hybrid elections broaden complexity-theoretic resistance to control. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1308–1314. AAAI Press, January 2007.

[HSV05] E. Hemaspaandra, H. Spakowski, and J. Vogel. The complexity of Kemeny elections. *Theoretical Computer Science*, 349(3):382–391, 2005.

[KP01] W. Kern and D. Paulusma. The new FIFA rules are hard: Complexity aspects of sports competitions. *Discrete Applied Mathematics*, 108(3):317–323, 2001.

[LLS75] R. Ladner, N. Lynch, and A. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1(2):103–124, 1975.

[McG53] D. McGarvey. A theorem on the construction of voting paradoxes. *Econometrica*, 21(4):608–610, 1953.

[MPRZ08] R. Meir, A. Procaccia, J. Rosenschein, and A. Zohar. The complexity of strategic behavior in multi-winner elections. *Journal of Artificial Intelligence Research*, 33:149–178, 2008.

[MPS08] J. McCabe-Dansted, G. Pritchard, and A. Slinko. Approximability of Dodgson's rule. *Social Choice and Welfare*, 31(2):311–330, 2008.

[MS97] V. Merlin and D. Saari. Copeland method II: Manipulation, monotonicity, and paradoxes. *Journal of Economic Theory*, 72(1):148–172, 1997.

[MT90] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley and Sons, 1990.

[MU95] I. McLean and A. Urken. *Classics of Social Choice*. University of Michigan Press, 1995.

[Pap94] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[PR07] A. Procaccia and J. Rosenschein. Junta distributions and the average-case complexity of manipulating elections. *Journal of Artificial Intelligence Research*, 28:157–181, February 2007.

[PRK07] A. Procaccia, J. Rosenschein, and G. Kaminka. On the robustness of preference aggregation in noisy environments. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 416–422. ACM Press, May 2007.

[PRVW08] M. Pini, F. Rossi, K. Venable, and T. Walsh. Dealing with incomplete agents' preferences and an uncertain agenda in group decision making via sequential majority voting. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning*, pages 571–578. AAAI Press, September 2008.

[RSV03] J. Rothe, H. Spakowski, and J. Vogel. Exact complexity of the winner problem for Young elections. *Theory of Computing Systems*, 36(4):375–386, 2003.

[Sat75] M. Satterthwaite. Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10(2):187–217, 1975.

[SM96] D. Saari and V. Merlin. The Copeland method I: Relationships and the dictionary. *Economic Theory*, 8(1):51–76, 1996.

[Ste59] R. Stearns. The voting problem. *The American Mathematical Monthly*, 66(9):761–763, 1959.

[Vaz03] V. Vazirani. *Approximation Algorithms*. Springer, 2003.

[Wal08] T. Walsh. Complexity of terminating preference elicitation. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems*, pages 967–974, May 2008.

[XC08] L. Xia and V. Conitzer. Determining possible and necessary winners under common voting rules given partial orders. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pages 196–201. AAAI Press, July 2008.

[XLY07] L. Xia, J. Lang, and M. Ying. Strongly decomposable voting rules on multi-attribute domains. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, pages 776–781. AAAI Press, July 2007.

[ZPR08] M. Zuckerman, A. Procaccia, and J. Rosenschein. Algorithms for the coalitional manipulation problem. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 277–286, January 2008.