AGH

Akademia Górniczo-Hutnicza
im. Stanisława Staszica
w Krakowie

Praca magisterska

The Complexity of Control Problems in Elections

Krzysztof Wojtas

Kierunek: Informatyka
Specjalność: Inżynieria systemów informatycznych i baz danych

Nr albumu: 203788

Promotor:
dr inż. Piotr Faliszewski

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki
Katedra Informatyki

Kraków 2011

Contents

Abstract

We study computational complexity of predicting winners in elections. We show the problem reduces to counting variants of control problems in elections. We consider several voting rules, such as: plurality, approval, Condorcet, and maximin. Regarding each of these methods we analyze altering the set of candidates by adding or deleting candidates, and altering the collection of voters in the same manner. We prove that each of these problems is either in FP or is #P-complete, and in the former case we provide a polynomial-time algorithm which solves it.

Keywords

election, voting, election system, social choice, plurality, approval, Condorcet, maximin, winner prediction, election control, computational complexity, FP, #P-complete

Chapter 1

Introduction

In this thesis we discuss some new results relating to computational aspects of elections. The main motivation of this work is to develop techniques of predicting winners in elections and we show this problem can be reduced to the counting variants of election control problems. For this reason we consider the computational complexity of the counting variants of certain types of control under several important voting systems. We formulate each problem we consider and for each problem and election system under consideration we provide a mathematical proof that either it can be solved in polynomial time, or that the problem is #P-complete and thus it is very unlikely that a fast solution exists. In the first situation, we also describe and construct an algorithm that solves the problem.

Some of the results from this thesis have been presented at IJCAI 2011 Workshop on Social Choice and Artificial Intelligence in [FW11].

1.1. What are Elections?

Elections are natural tools when a group of people—or agents in a multi-agent system—must decide which option to choose from a given set of alternatives. The group may contain a large number of participants, who typically disagree which alternative(s) are more desirable than others. This requires the group to aggregate the preferences of the individuals to reach a decision which is acceptable to the members of the society as a whole. Often, these preference aggregation systems involve some form of voting, and such process is particularly important in this work. We will call the members of the group that participate in the voting *voters*, and we will call the alternatives from which they choose *candidates*.

The main purpose of an election is the determination of—informally speaking—the most acceptable candidate, which is termed as a *winner* of the election. In some circumstances there is not only a single winner but a group of winners, each equally appropriate.

In an ideal world elections should be fair. Every participant of an election should be able to vote for a candidate of their choice, and each single vote should be worth the same. Thus, groups use *voting systems*, in which the winner(s) are determined in an algorithmic manner. We will now informally introduce some key concepts of the model we will use. First, in some fashion a set of valid candidates is generated. Each voter is given the list of candidates and reports his preferences over these candidates. In almost all the elections we will discuss, a vote is simply an ordered list of all the candidates from the most desirable one to the least desirable one. Once the preferences are collected, some rule is used to combine them into a final result, i.e., a set of one or more winners or even no winners at all. The empty set of winners may happen in some special cases, e.g., when every candidate gains the same rate of support from the voters and no one is clearly the leader.

There are many different voting systems, each coming with different advantages and disadvantages. Probably the most common one is plurality voting, in which voters are asked only for their first preference, and the alternative which receives the greatest number of votes wins. One can also consider veto voting, which can be viewed as a dual system to plurality—each voter specifies the least desirable candidate and the winner is the one who gained the least number of votes. Other common systems include the single transferable vote (or instant runoff), which repeatedly eliminates the candidate with the least number of first preferences, and reassigns the votes to the voters' next-highest preferences, until only the desired number of winners is left. Another system is approval voting, in which, rather than ranking candidates, voters indicate whether or not they approve of each candidate. The winner(s) are the candidate(s) with the highest number of voters approving of them. In the Condorcet voting a candidate is a winner exactly if he beats each other candidate in a head-to-head contest under the voters' preferences. This means that in order to be a winner, a candidate must be preferred to each other candidate by more than half of the voters (however, not necessarily the same voters for each of the candidates). Thus, there can be a situation in which we do not have a Condorcet winner.

The mystic and philosopher Ramon Llull defined the Llull system in the 13th century. In this system one considers each pair of candidates and awards one point to the winner of the head-to-head contest, and if the head-to-head contest is a tie, each still gets one point. A similar voting scheme to Llull is Copeland, in which one also performs head-to-head contest between each pair of candidates, and in case of a tie each candidate of the pair gets half a point. The Llull/Copeland system is used in the group stage of the World Cup soccer tournament, except there (after rescalling) wins get one point and ties get one third of a point.

We have chosen several well-known and well-studied voting systems, and in this work we examine their computational complexity regarding some problems on elections. The formal definitions of the voting systems, as well as those regarding problems in elections, will be presented in Chapter 3.

1.2. Applications of Voting

It turns out that a large number of situations (both theoretical and practical) can be described in terms of voting. The most obvious examples are political elections which are used as a tool for selecting representatives in democratic countries or other administrative units. In many countries presidential and parliamentary elections are performed every few years and millions of people choose one person or party from several possibilities. Also, within states, regions, or provinces, local government authorities are chosen regularly and by their structure these elections resemble presidential elections, although in this case there are significantly fewer voters.

In political elections it may often be the case that voters are not informed about every candidate, or feel relatively neutral about their "middle" preferences. In particular, they may strongly like or dislike only one, or a few candidates. In these cases protocols such as plurality or veto voting may be reasonable models.

Very similar to the political elections are those performed in companies, organizations, clubs, and by other groups of people, both large and small, formal and informal. Members of such groups can elect a new leader, a new budget plan, a new logo, new company objectives (e.g., whether or not to merge with another company), etc. Compared with the political elections, there are still a few possibilities to choose but the number of voters does not exceed several thousand; in small groups there can be even below a hundred of them.

As we can see, many important decisions involve elections and voting, but they are widely used not only in high-impact situations, but also in more frivolous ones. Suppose a radio station needs to determine the most popular song of the month. A good idea is to organize a contest where listeners can vote on songs (e.g., via text messages) from a list of recent ones. This list can be fixed by the organizers, or there can be a possibility to modify the list by adding new songs during the contest if it turns out that a large number of people like them and want to promote them. One can also imagine situations in which the list of songs is fixed, but every day after ranking the songs by the listeners, the song with the lowest score is rejected. The one which remains on the list until the end of the contest becomes the most popular song of the month. The elimination of the alternatives is also popular in game shows—every series of the show introduces a new group of contestants and in every subsequent episode of the series the next contestant is eliminated. In these examples of voting the set of candidates is not fixed. In fact, neither is the set of voters, since one particular person may not vote in each round of the elimination. From the point of view of theory of elections, this kind of voting is interesting since we must allow for both sets to vary.

In the previous examples the set of voters and/or the set of candidates consist of humans. Also, there were always significantly more voters than candidates. However, one can consider systems in which both voters and candidates are not humans but intelligent software agents that learn and use knowledge to achieve their goals. In these settings, it is reasonable to speak of an election of arbitrary numbers of candidates (and perhaps relatively few voters). For example, a financial professional may have to decide between thousands of possible stock purchases (candidates), each of which may be ranked differently by each of dozens or hundreds of experts and computer programs (voters).

Using voting theory, Ephrati and Rosenschein [ER93] present a new approach to deriving multi-agent plans. They consider the situation where autonomous agents attempt to coordinate action and how they could reach consensus using a voting mechanism. The authors introduce a new multi-agent planning technique that makes use of a dynamic, iterative search procedure. Each agent expresses its preferences, and a group choice mechanism is used to select the result. At each step, agents vote about the next joint action in the group plan (i.e., what the next transition state will be in the emerging plan). Using this technique agents need not fully reveal their preferences, and the set of alternative final states need not be generated in advance. Since each agent has its own private goal and all agents operate in the same environment and may share resources, it is desirable that they agree on a joint plan for the group that will transform the world to an agreed final state.

In the context of the Web, the main applications of voting include building meta-search engines, combining ranking functions, selecting documents based on multiple criteria, and improving search precision through word associations. For example, Dwork et al. [DKNS01] propose the use of voting to combine search results from various sources from the Web. Given the particular query, one creates an election in which every Web page from the search space acts as a candidate, and every search engine acts as a voter. Note that in contrast to the social choice scenario where there are many voters and relatively few candidates, in the Web aggregation scenario there are many candidates and relatively few voters. The authors develop a set of techniques that involve the model of voting for the rank aggregation problem and compare their performance to that of well-known methods. A primary goal of their work is to design rank aggregation techniques that can simply, efficiently and effectively combat "spam", which is a serious problem in Web searches.

Recommender systems apply data mining techniques and prediction algorithms to predict users' interest in knowledge, products, and services among the tremendous amount of available items. The vast growth of the amount of information on the Internet as well as the number of visitors on websites add some serious challenges to recommender systems, such as producing accurate recommendation, handling many recommendations efficiently and coping with the vast growth of number of participants in the system. Therefore, new recommender system technologies are needed that can quickly produce high quality recommendations even for huge data sets. For example, Ghosh et al. [GMHS99] designed an automated movie recommendation system, in which the conflicting preferences a user may have about movies were represented as agents, and movies to be suggested were selected according to a voting scheme (in this example there are multiple winners, as several movies are recommended to the user).

As one could notice, elections appear in many real-world applications, thus it is clear they are important not only for humans, but they are also often used in computer environments. In the next section we take a look at the challenges concerning elections and how we can deal with them using computational complexity theory.

1.3. Problems with Elections

There are many interesting algorithmic problems that arise when we study elections. As we could see in the previous section, the election outcomes often involve important matters such as who will be the next political leader or how huge amounts of money will be spent. Thus, it is reasonable to assume that in some cases individuals will attempt to interfere with voting procedures, or will vote dishonestly, in order to gain a personal advantage or to have more influence on the result. The three main types of attacks, studied within computational social choice literature, aimed on altering an election in order to change its outcome are: control, manipulation, and bribery. We discuss them in the next paragraphs.

The situation where election authorities may alter the election structure to change the results, without changing how voters choose to vote, is called *control.* In election control some actor (traditionally called “the chair”) who has complete knowledge of all the votes seeks to achieve a desired outcome via changing the structure of the election. There are several varieties of control problems; those proposed by Bartholdi, Tovey, and Trick [BTT92] are: adding or deleting candidates, adding or deleting voters, and partitioning candidates or voters into a 2-round election structure. When attempting to control, one could try to make a particular candidate become the winner of an election or to make a certain candidate lose an election. The former way, introduced in [BTT92], is called *constructive case*, and the latter is called *destructive case.* The destructive cases of control problems have been defined and studied by Hemaspaandra, Hemaspaandra, and Rothe [HHR07] under several voting systems. We will describe the types of control that we focus on in greater detail in Chapter 3.

As a real-life example, consider the 2000 U.S. Presidential election. If one wanted George W. Bush to win (and Ralph Nader was not at that time running), he or she might have chosen to add the candidate Ralph Nader which would split votes away from Al Gore and achieve one’s desired outcome. Bush defeated Gore by 537 votes. Nader received 97,421 votes, which led to claims that he did play a pivotal role in determining who would become the president.

If individual voters or some coalition of voters (termed manipulators) cast votes which do not honestly represent their preferences, but which lead to a more favorable outcome (e.g., in which a designated candidate is a winner), we refer to this as *manipulation.* Like control, this problem comes in two versions: constructive and destructive. One can also consider the model in which each voter has a non-negative integer weight determining his importance in an election. It turns out, after Gibbard-Satterthwaite theorem [Gib73, Sat75], that every voting system can be manipulated, unless there is some candidate who can never win, or the system is dictatorial (i.e., there is a single individual who can choose the winner).

We describe an example scenario which emphasizes what is manipulation capable of. Consider an election in which voters vote on three options, say a, b and c, and they are split quite evenly between a and b, with a small minority of voters who prefer c most. Nonetheless, about half of the voters prefers a to b and b to c (which we denote $a \succ b \succ c$), and about half are the other way round—they prefer b to a and a to c ($b \succ a \succ c$). The voting rule we use is called Borda count: We give 2 points to our top choice, 1 point to our middle, and 0 points to the worst. In order to make candidate a a winner, some of his supporters could assume the following strategy: Why vote $a \succ b \succ c$

and give the point to b, when c is not winning anyway? So they change their preferences to be $a \succ c \succ b$. But in effect candidate c collects enough points to beat both a and b which was clearly not the manipulators' goal.

Finally, it is possible that an alteration of how voters vote is accomplished by bribing the voters. *Bribery*, as defined by Faliszewski, Hemaspaandra, and Hemaspaandra [FHH09a], is a natural extension of manipulation. We assume the actor performing the attack—called briber—has a certain budget, and the voters each have a price for which their vote can be bought. The question is whether the briber can make a preferred candidate be a winner by buying the votes and altering them appropriately while not exceeding the budget. Like some types of control one has to choose which collection of voters to act on, but like manipulation one is altering votes.

When we go for shopping we act as voters when choosing from a variety of similar products. It is common practice when a producer sells their products with some gadgets inside to attract customers and make them choose the particular brand. Because those gadgets are usually for free and customers pay only for the product itself, more of these products are sold and their trademark is therefore well-remembered by customers. It turns out we are bribed almost every day.

Bartholdi, Tovey, and Trick [BTT92, BTT89a] initiated a line of research whose goal is to protect elections from various attacking actions intended to change their results. Their strategy for achieving this goal was to prevent attacking by using computational complexity. They show that for various election systems and attacking actions, even seeing whether for a given collection of votes an attack is possible is NP-complete. Such systems are referred to as *resistant*. Along with *vulnerable* systems that can be attacked using polynomial-time algorithms, they form a class of *susceptible* election systems. Every susceptible system can be attacked given enough time.

It would be ideal if we could design voting systems that are *immune* to described types of attacks (which means an attack would be impossible to accomplish), or at least that such systems would be resistant to many types of attacks. For example, Copeland and Llull systems have a great number of resistances to constructive control [FHHR09a]. Hemaspaandra, Hemaspaandra, and Rothe [HHR09] constructed a hybrid voting system which is resistant to every control type mentioned in the previous paragraphs. However, such hybridization is artificial and impractical for most applications. Thus, at best we can seek to find a voting system, for which such attacks are either impossible to accomplish or computationally infeasible to determine how to accomplish, only in certain cases.

Many voting schemes have been considered to see their susceptibility and resistance to various types of control. Hemaspaandra, Hemaspaandra, and Rothe [HHR07] study approval voting and destructive versions of plurality and Condorcet. Control and bribery properties of Llull and Copeland systems are discussed by Faliszewski et al. [FHHR09a]. Tideman [Tid87], and Elkind, Faliszewski, and Slinko [EFS10] also consider such tricks as candidate cloning. In this model a manipulator can replace each candidate by one or more clones in order to make its most preferred candidate (or one of its "clones") win the election. Multimode control attacks, i.e., a single control attack with multiple standard types of control combined, are studied by Faliszewski, Hemaspaandra, and Hemaspaandra [FHH11b]. Manipulation has been studied under numerous voting systems, in both constructive and destructive situations, and where the voters are weighted. The study of the complexity of manipulation for the unweighted case for some voting systems was initiated by Bartholdi, Tovey, and Trick [BTT89a] and was continued by Conitzer, Lang, and Sandholm [CSL07] who discuss how many candidates it takes for manipulation to become NP-complete for various voting systems in the weighted case. In [HH07] Hemaspaandra and Hemaspaandra present a dichotomy theorem regarding the complexity of manipulation for a class of voting rules called scoring protocols. Faliszewski, Hemaspaandra, and Hemaspaandra [FHH09b][1] initiated the study of bribery from a computational complexity point of view. They find the complexity of bribery for many variations, such as weighted voters and varying voters' prices, and under a number of voting systems. We point the reader to the works of Faliszewski et al. [FHH10], and Faliszewski and Procaccia [FP10], which are general references on the subject of the complexity of manipulating elections.

Many other problems involving elections are not focused on attacking them. For example, even the winner determination can be a non-trivial challenge that in some circumstances has been proven not to be solvable in polynomial time (under standard complexity theoretic assumptions such as $P \neq NP$). In other words, for some specific voting systems given all parameters of the particular election, there is no fast algorithm which will tell us the outcome of the election. That means that for many practical applications described in the previous section, due to large sizes of voter set and/or candidate set it is next to impossible to tell what the result of an election is.

In this work we do not study such bizarre cases. However, we point the readers to the papers of Faliszewski et al. [FHHR09b] and [FHH10]. The work of Bartholdi, Tovey, and Trick [BTT89b] is one of the first pieces of research on this subject in which the authors discuss the winner determination problem and show its NP-hardness for two voting schemes: Dodgson and Kemeny. Dodgson voting system is further studied by Hemaspaandra, Hemaspaandra, and Rothe [HHR97] with a result that the winner determination is Θ_2^p-complete. Hemaspaandra, Spakowski, and Vogel [HSV05] proved Θ_2^p-completeness for the winner problem in Kemeny elections, and Rothe, Spakowski, and Vogel [RSV03] proved the same for the winner problem in Young elections.

1.4. Winner Prediction Under Uncertainty

We will now discuss one more problem, which is the particular motivation for this work. It focuses on some type of uncertainty in elections. Such models have been considered in many ways, two significant examples are *possible winner* and *necessary winner* problems introduced by Konczak and Lang [KL05] and further studied by many other researchers; see, e.g., Xia and Conitzer [XC08]; Betzler and Dorn [BD09]. These problems focus on voting schemes with incomplete knowledge about voters' preferences and answer the question if a candidate is a winner for some extension of preference orders (possible

1. The conference version of this paper is from 2006.

winner), or if he is a winner no matter how we extend preference orders (necessary winner).

Our problem focuses on a different type of uncertainty—we consider elections where the set of participating candidates and the set of voters are uncertain. In the full generality of this problem we do not know which candidates will take part in the election and which voters will cast their votes. However, for each subset of candidates we know the probability that exactly candidates in this subset participate in the election, and for each subset of voters we know the probability that exactly voters in such subset cast their votes. Our goal is to compute the probability that a given candidate wins the election. The problem can very easily become computationally hard with unrestricted probability distributions. Therefore we provide some conditions that should hold in these distributions, so that the large class of cases become computationally easy. The formal description of the winner prediction model along with those restrictions is presented in Section 3.3.

It turns out that our winner prediction problem (after applying the restrictions) can be reduced to counting variants of election control problems. We focus on several popular voting systems and we ask how hard it is to not only find whether such type of control is possible, but to find out how many such possibilities exist. Our results show that counting variants of control problems under plurality, approval and Condorcet systems are polynomial-time solvable whenever the decision variants are. (Note that this is far from obvious: It is well-known that deciding if a perfect matching in a bipartite graph exists is polynomial-time computable, but counting such matchings is #P-hard, [Val79].) This means that for the respective cases our winner prediction problems are polynomial-time solvable. Only in maximin we conjecture there is a difference between counting and decision variants—we believe that the case of deleting candidates is computationally hard when we count solutions while it is computationally easy when we decide if a solution exists.

1.5. The Structure of the Thesis

This thesis has the following structure. In Chapter 2 we provide brief background of mathematical foundations and a quick recapitulation of the most essential notions of computational complexity theory. The latter is used in descriptions of algorithms and proofs we construct, and the former is needed to define some of the more complex structures such as elections and voting systems. We describe elections and voting systems in Chapter 3 along with precise definitions of our control problems and a formal description of prediction of the winner model. Further chapters are the core of this work—they describe results for individual voting system. They are: plurality voting in Chapter 4, approval voting and Condorcet voting both in Chapter 5, and maximin voting in Chapter 6. We conclude in Chapter 7 with summarizing remarks and suggestions for further research.

Chapter 2

Preliminaries

In order to formally study elections, we need to use a precise mathematical apparatus. In Section 2.1 we define certain set-theoretic structures that we use in the further work and that need a subtle treatment. In Section 2.2 we define some useful notions of formal language theory and computational complexity theory. Throughout this chapter we also provide brief examples that help understand the theory we introduce. An experienced reader can either skim or skip this chapter. However, our definitions may slightly differ from those in literature. Also, we make some assumptions that will be used in the whole work and that are described here.

2.1. Basic Concepts

We start by defining some basic mathematical notions. We assume that the reader is familiar with such fundamental set-theory notions as set, union, intersection, difference, subset, empty set, set membership, and ordered pair.

Let us define the set of naturals to be $\mathbb{N} = \{0, 1, \dots\}$ (that is, we take $\mathbb{N}$ to contain 0) and the set of integers to be $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$.

Definition 2.1. *A Cartesian product of two sets A and B, denoted $A \times B$, is a set of all ordered pairs whose first element is a member of A and whose second element is a member of B:*

$$A \times B = \{\, (a, b) : a \in A,\ b \in B \,\}.$$

Product $A \times A$ is often denoted A^2.

Definition 2.2. *A (binary) relation R between sets A and B is a subset of $A \times B$. If $A = B$, we say that R is a relation over A. If $(a, b) \in R$, we denote this by $a\,R\,b$.*

Example 2.1. Suppose we have the set of card figures

$$F = \{\mathit{Ace}, \mathit{Two}, \mathit{Three}, \mathit{Four}, \mathit{Five}, \mathit{Six}, \mathit{Seven}, \mathit{Eight}, \mathit{Nine}, \mathit{Ten}, \mathit{Jack}, \mathit{Queen}, \mathit{King}\},$$

and the set of card suits

$$S = \{\mathit{Spades}, \mathit{Clubs}, \mathit{Hearts}, \mathit{Diamonds}\}.$$

The set $F \times S$ represents the complete 52-card deck. Every 5-element set R, $R \subseteq F \times S$, is a relation between sets F and S representing a single poker hand. $\square$

Definition 2.3. *We say that a relation R over A is:*

1. *reflexive, if for all $a \in A$ it holds that $a\,R\,a$;*
2. *symmetric, if for all a, $b \in A$ it holds that $a\,R\,b$ implies $b\,R\,a$;*
3. *antisymmetric, if for all a, $b \in A$ it holds that $a\,R\,b$ and $b\,R\,a$ implies $a = b$;*
4. *transitive, if for all a, b, $c \in A$ it holds that $a\,R\,b$ and $b\,R\,c$ implies $a\,R\,c$;*
5. *total, if for all a, $b \in A$ it holds that either $a\,R\,b$ or $b\,R\,a$.*

Definition 2.4. *Let A be some set and let $\succ$ be some relation over A. Relation $\succ$ is called a partial order, if it is reflexive, antisymmetric and transitive. If $\succ$ is also total, we say that $\succ$ is a total (or linear) order.*

Note for a partial order $\succ$ over A that there can be such elements a, $b \in A$ for which neither $a \succ b$ nor $b \succ a$ holds, i.e., these elements are incomparable. If the relation $\succ$ is a total order, such elements do not exist.

We can now define functions. Informally speaking, a *function* is a relation between sets A and B that binds every element from A with exactly one element from B.

Definition 2.5. *Let A, B be some sets, and let f be a relation between A and B. We say that f is a function if:*

1. *for all $a \in A$ there exists $b \in B$ such that $(a, b) \in f$, and*
2. *for all $a \in A$, b_1, $b_2 \in B$ it holds that $(a, b_1) \in f$ and $(a, b_2) \in f$ implies $b_1 = b_2$.*

If f is a function between A and B, we write $f\colon A \mapsto B$. If $(a, b) \in f$, we denote this by $f(a) = b$.

Definition 2.6. *Let A be some set and let f be a function, $f\colon A \mapsto \mathbb{N}$. An ordered pair $X = (A, f)$ is called a multiset over A. We write $a \in X$, if $a \in A$ and $f(a) > 0$.*

Multisets are generalizations of sets. Intuitively, every element of a multiset is paired with an information regarding how many copies of it are members of the multiset. The first element of the pair, A, from the definition above denotes the set from which we take elements, and the second element, called multiplicity function, for every $a \in A$ assigns its multiplicity (i.e., the number of instances of this element present in the multiset). If we do not want to specify at the given moment if the structure we are speaking of is a set or a multiset, we use a word *collection*.

Let us now define basic operations on multisets.

Definition 2.7. *Let $X = (A, f)$ and $Y = (A, g)$ be multisets over A. Their sum, denoted $X \uplus Y$, is the multiset $Z = (A, h)$, where for all $a \in A$, $h(a) = f(a) + g(a)$.*

The symbol $\uplus$ is used here to denote the sum of multisets, as opposed to their union ($\cup$) defined further. It can be easily shown that the multiset sum operation has the following properties:

1. commutativity: $X \uplus Y = Y \uplus X$,
2. associativity: $X \uplus (Y \uplus Z) = (X \uplus Y) \uplus Z$, and
3. there exists the null multiset $\emptyset$, such that $X \uplus \emptyset = X$.

It is important to note that the multiset sum is not idempotent, i.e., $X \uplus X \neq X$.

Definition 2.8. *Let $X = (A, f)$ and $Y = (A, g)$ be multisets over A. Their union, denoted $X \cup Y$, is the multiset $Z = (A, h)$, where for all $a \in A$, $h(a) = \max(f(a), g(a))$.*

Definition 2.9. *Let $X = (A, f)$ and $Y = (A, g)$ be multisets over A. Their intersection, denoted $X \cap Y$, is the multiset $Z = (A, h)$, where for all $a \in A$, $h(a) = \min(f(a), g(a))$.*

Definition 2.10. *Let $X = (A, f)$ and $Y = (A, g)$ be multisets over A. Their difference, denoted $X - Y$, is the multiset $Z = (A, h)$, where for all $a \in A$, $h(a) = \max(0, f(a) - g(a))$.*

Unlike the sum of multisets, the operation of multiset intersection and the operation of multiset difference can be viewed as generalizations of plain set intersection and plain set difference. Thus, the usage of the same symbols does not lead to misunderstandings. Well-known laws involving unions, intersections and differences that hold for sets, are also valid when regarding multisets.

Definition 2.11. *Let $X = (A, f)$ and $Y = (A, g)$ be multisets over A. We write $X \subseteq Y$ if, for all $a \in A$, $f(a) \leq g(a)$. We say multisets X and Y are equal, if $X \subseteq Y$ and $Y \subseteq X$, which holds only if $f = g$.*

We use the same method for describing elements of some multiset as it is done for sets. For example, by writing $X = \{1, 1, 2\}$ we mean the multiset X that contains two copies of 1, and one copy of 2. However, we should say what the underlying set of the multiset is beforehand. This is important, because X from this paragraph can be viewed as $X = (A, f)$ such that, for example:

(i) $A = \{1, 2\}$ and $f(1) = 2$, $f(2) = 1$, or

(ii) $A = \mathbb{N}$ and $f(1) = 2$, $f(2) = 1$, and for all $a \in \mathbb{N} - \{1, 2\}$, $f(a) = 0$.

(There are, of course, more interpretations of this notation than these two.) Thus, it is necessary to indicate what A is. We should also remember that, unlike for sets, $\{1, 1\} \neq \{1\}$.

Definition 2.12. *Let A be some set. The cardinality of A, denoted by $|A|$, is the number of elements in A. Set A is finite if it has a finite number of elements. Set A is empty if $|A| = 0$.*

Definition 2.13. *Let $X = (A, f)$ be some multiset over set A. The cardinality of X, denoted by $|X|$, is the number $\sum_{a \in A} f(a)$. Multiset X is finite if set A is finite. Multiset X is empty if $|X| = 0$.*

In this work most of sets and multisets we consider are finite, unlike $\mathbb{N}$ which is an *infinite* set. Particularly, every election we study in this work consists of a finite set of candidates and a finite multiset of voters.

Example 2.2. Let $X = (A, f)$ and $Y = (A, g)$ be multisets over $A = \{1, 2, 3\}$ such that $X = \{1, 1, 2\}$ and $Y = \{1, 3\}$. We have:

$$\begin{aligned} X \uplus Y &= \{1, 1, 1, 2, 3\}; \\ X \cup Y &= \{1, 1, 2, 3\}; \\ X \cap Y &= \{1\}; \\ X - Y &= \{1, 2\}. \end{aligned}$$

It is true that $\{1, 1\} \subseteq X$, but $\{1, 1\} \nsubseteq Y$. The cardinalities of X and Y are: $|X| = 3$, $|Y| = 2$. □

We will need some basic notions of graph theory, in particular a directed graph and an induced subgraph.

Definition 2.14. *A (directed) graph G is an ordered pair (V, A) where V is a finite set of vertices and $A \subseteq V \times V$ is a set of arcs.*

If $G = (V, A)$ is a graph, $a \in A$, and $a = (u, v)$, we say that a is an arc from u to v or that a is *leaving* u and a is *entering* v. Often we modify graphs by adding or removing vertices or arcs. These operations are straightforward, however, when we attempt to remove vertices from a graph, we automatically remove each arc that leaves or enters the removed vertices. By removing vertices we obtain induced subgraphs.

Definition 2.15. *Let $G = (V, A)$ be a directed graph and $V' \subseteq V$. We say $G' = (V', A')$ is an induced subgraph of G, if $A' = A \cap (V' \times V')$.*

Let us now define some notations we will need to describe the asymptotic running time of algorithms.

Definition 2.16. *Let g be a function, $g\colon \mathbb{N} \mapsto \mathbb{Z}$. We denote by $\Theta(g(n))$, $O(g(n))$, $\Omega(g(n))$ the following sets of functions:*

$$\begin{aligned} \Theta(g(n)) = \{\, f\colon \mathbb{N} \mapsto \mathbb{Z} : {}& \textit{there exist positive constants } c_1,\ c_2, \textit{ and } n_0 \textit{ such that} \\ & 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \textit{ for all } n \ge n_0 \,\}, \\ O(g(n)) = \{\, f\colon \mathbb{N} \mapsto \mathbb{Z} : {}& \textit{there exist positive constants } c, \textit{ and } n_0 \textit{ such that} \\ & 0 \le f(n) \le c g(n) \textit{ for all } n \ge n_0 \,\}, \\ \Omega(g(n)) = \{\, f\colon \mathbb{N} \mapsto \mathbb{Z} : {}& \textit{there exist positive constants } c, \textit{ and } n_0 \textit{ such that} \\ & 0 \le c g(n) \le f(n) \textit{ for all } n \ge n_0 \,\}. \end{aligned}$$

Because $\Theta(g(n))$ is a set, we could write $f \in \Theta(g(n))$ to indicate that f is a member of $\Theta(g(n))$. Instead, by a slight abuse of notation, we will typically write $f = \Theta(g(n))$ to express the same notion (the same rule applies to notations O and Ω). If $f = \Theta(g(n))$, we say that function g is *asymptotically tight bound* for f. If $f = O(g(n))$, we say that function g is *asymptotically upper bound* for f. If $f = \Omega(g(n))$, we say that function g is *asymptotically lower bound* for f.

From the definition of the asymptotic notations, it is easy to prove the following proposition.

Proposition 2.1. *For any two functions f, $g\colon \mathbb{N} \mapsto \mathbb{Z}$, we have $f = \Theta(g(n))$ if and only if $f = O(g(n))$ and $f = \Omega(g(n))$.*

Example 2.3. Let $f(n) = n - 5$ and $g(n) = 2011n^2 - 2012n + 1$. We have:

$$\begin{array}{llllll} f = O(n^2) & \text{and} & f \neq \Omega(n^2) & \text{and thus} & f \neq \Theta(n^2); \\ g = O(n^2) & \text{and} & g = \Omega(n^2) & \text{and thus} & g = \Theta(n^2); \\ f = O(n^2) & \text{and} & g = O(n^2) & \text{but it does not mean that } f = g. \end{array}$$

□

2.2. Computational Complexity Review

We seek algorithmic and complexity-theoretic results regarding various election-related problems. Thus, we now review basic notions of computational complexity theory.

We start by giving formal definitions of what function and decision problems are from the point of view of complexity theory.

Definition 2.17. *A (computational) problem Π is a binary relation between a set $\mathcal{I}$ of problem instances and a set $\mathcal{S}$ of problem solutions. If the binary relation is a function, we call Π a function problem.*

Definition 2.18. *A decision problem Π is a function between a set $\mathcal{I}$ of problem instances and a set $\{\text{“yes”}, \text{“no”}\}$.*

In computational problems there can be many correct outputs for the specified input, while in function problems, given an instance, there is a single correct answer. Decision problems focus on determining whether for some their instances there is a solution or not, without finding the particular solution. Of course, every decision problem is a function problem, and every function problem is a computational problem.

Example 2.4. Let us consider the computational problem of determining any prime divisor of a natural number greater than 1. This problem can be viewed as a binary relation $\Pi \subseteq \mathcal{I} \times \mathcal{S}$, such that $\mathcal{I} = \mathbb{N} - \{0, 1\}$ and $\mathcal{S}$ is the set of all primes. It holds, e.g., $(5, 5) \in \Pi$, $(12, 2) \in \Pi$, $(12, 3) \in \Pi$, $(12, 5) \not\in \Pi$. Since there are numbers that have more than one prime divisor (e.g., 12), Π is not a function problem.

Let us consider another computational problem of determining whether a natural number is prime or not. It can be viewed as a function $\Pi'\colon \mathcal{I}' \mapsto \{\text{“yes”}, \text{“no”}\}$, such that $\mathcal{I}' = \mathbb{N}$. Thus, Π' is a decision problem. It holds that, e.g., $\Pi'(5) = \text{“yes”}$, $\Pi'(12) = \text{“no”}$.

□

If a computer program is to solve a computational problem, problem instances must be represented in a way that the computer understands. An *encoding* for a problem Π provides a way of describing each instance of Π by an appropriate string (a finite sequence) of symbols over some fixed set, like $\{0, 1\}$. For example, we are familiar with encoding b of the natural numbers $\mathbb{N}$ as the binary strings $\{0, 1, 10, 11, 100, \dots\}$. Using this encoding, $b(13) = 1011$. Anyone who has looked at computer representations of keyboard characters is familiar with either the ASCII or EBCDIC or Unicode codes. In the ASCII code, the encoding of `A` is 1000001. A compound object, that is, an object made of a number of subobjects, can be encoded as a binary string by combining the representations of its constituent parts. Polygons, graphs, functions, ordered pairs, programs—all can be encoded as binary strings. Thus, every problem can be transformed to the one whose every instance is the encoded instance of the original problem.

Of course, some of encodings are better than others. Generally, good encodings used for a particular type of object should be concise and decodable. The former means that when encoding instances of a problem we should not use any unnatural "padding" that could be used to artificially expand the length of an instance. The intent of the latter is that we should be able to specify fast method for transforming the encoded instance to the instance itself. In particular, encoding natural numbers in binary is suitable, since we can efficiently transform the encoded string of zeros and ones to the number it represents, and there are no "paddings" in the string. On the other hand, the unary encoding u (a natural number n is then represented by a sequence of $n+1$ ones) would be problematic, since the transformation between n and $u(n)$ requires n steps. Such transformation is much slower than the one which converts n to its binary representation in $\log_2 n$ steps. Logarithmic functions increase much slower than linear functions, thus representing numbers in binary is more reasonable. Our goal is to present an encoding-independent approach to the theory of the computational complexity, but we assume that we use only the good encodings for our problems. For an extended discussion about encoding schemes see Garey and Johnson [GJ90, pp. 18–23].

Definition 2.19. *A computational problem Π is a concrete problem, if the set of its instances $\mathcal{I}$ is the set of binary strings.*

In order to define some fundamental notions of computational complexity, we will now focus only on concrete decision problems. One of the advantages of this restriction is the ease of use of concepts of the formal language theory. Formal languages are very natural, formal counterparts of concrete problems, which are suitable objects to study in a mathematically precise theory of computation. They are defined in the following way.

Definition 2.20. *An alphabet Σ is a finite set of symbols. A string is a sequence (finite or not) of symbols from an alphabet. A language L over Σ is a set of strings made up of elements from Σ. We denote the empty string by ε, and the empty language by $\emptyset$. The language of all strings over Σ is denoted Σ^*.*

Example 2.5. Let us consider an alphabet $\Sigma = \{0, 1\}$. The set

$$L = \{10, 11, 101, 111, 1011, 1101, 10001, \dots\}$$

is the language over Σ containing prime numbers written in binary, and the set

$$\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$$

is the language over Σ containing every finite binary sequence. Every language over Σ is a subset of Σ^*. $\square$

Definition 2.21. *Let Σ be an alphabet and let $x \in \Sigma^*$ be a string. The length of x, denoted $|x|$, is the number of symbols in x.*

Example 2.6. We have $|001101| = 6$, $|1| = 1$, and $|\varepsilon| = 0$. □

From the point of view of the formal language theory, the set of instances for any concrete decision problem Π is simply a subset of Σ^*, where $\Sigma = \{0, 1\}$. Since Π is entirely characterized by those problem instances that produce a "yes" answer, we can view Π as a language L over $\Sigma = \{0, 1\}$, where

$$L = \{\, x \in \Sigma^* : \Pi(x) = \text{“yes”} \,\}.$$

If a result holds for formal language L, then it holds for problem Π.

Example 2.7. Let us consider the problem Π' from Example 2.4, i.e., the problem of determining whether a natural number is prime or not. We use binary encoding b to transform every instance n of Π' to a binary string $b(n)$, and in effect we create a concrete problem Π''. The language corresponding to Π'' is

$$\text{PRIMES} = \{\, b(n) \in \{0, 1\}^* : n \text{ is prime}\}.$$

□

In order to formalize our notion of an algorithm, we will need to fix a particular model for computation. The model we will describe is the *deterministic Turing machine* (DTM). It consists of:

1. a two-way infinite tape made of cells containing symbols from a given alphabet,
2. a head that can read and write symbols in the tape's cells and move the tape left and right one cell at a time, and
3. a finite state control responsible for controlling the machine; this mechanism, given the state the machine is currently in and the symbol it is reading on the tape (the symbol currently under the head), tells the machine to do the following in sequence: either erase or write a symbol in the current cell, move the head, assume the same or a new state.

DTMs are not intended as practical computing technology, but can be adapted to simulate the logic of any computer algorithm. We will now give a formal definition of a DTM.

Definition 2.22. *A deterministic Turing machine is a 5-tuple $\mathcal{M} = (Q, \Gamma, \Sigma, \delta, q_0)$, where:*

1. *Q is a finite, non-empty set of states,*
2. *Γ is a work alphabet containing $\rhd$ (the final symbol) and $\sqcup$ (the blank symbol),*
3. *$\Sigma \subseteq \Gamma - \{\rhd, \sqcup\}$ is an input alphabet,*
4. *$\delta \colon Q \times \Gamma \mapsto \Gamma \times \{\leftarrow, \rightarrow\} \times (Q \cup \{\textsc{halt}, \textsc{yes}, \textsc{no}\})$ is a transition function, and*
5. *$q_0 \in Q$ is the initial state.*

In addition, we assume that $Q \cap \Gamma = \emptyset$ and $\leftarrow, \rightarrow, \textsc{halt}, \textsc{yes}, \textsc{no} \notin \Gamma$, and for each $p, q \in Q$, if $\delta(p, \rhd) = (s, d, q)$ then $s = \rhd$ and $d = \rightarrow$.

We will now discuss this definition and show how a DTM works. Let us assume that the machine $\mathcal{M}$ was executed on the string $x \in \Sigma^*$, called the *input*, that was stored in the consecutive cells of the tape, symbol by symbol. Directly to the left of the x's first symbol there is a cell containing the symbol $\rhd$, and the other cells on the tape contain the symbol $\sqcup$. The machine starts working by setting the head to the cell containing the first symbol of x and by assuming the initial state q_0. Then it performs the series of steps according to the transition function δ. During each step, if the machine is in a state $p \in Q$, the symbol in the current cell (i.e., the cell under the head) is $a \in \Sigma$, and $\delta(p, a) = (b, d, q)$, then it performs the following operations:

1. it writes the symbol b in the current cell (the new symbol erases the old one, and writing the blank symbol $\sqcup$ is plain erasing the cell's content),
2. it moves the head to the direct cell on the left (if $d = \leftarrow$) or to the direct cell on the right (if $d = \leftarrow$), and
3. it sets a new state q.

The restriction of the function δ from the definition of DTM prevents the head from writing on the cell with $\rhd$ and to the left of it. The machine stops if, after the finite number of steps, it is in the state HALT, YES, or NO. If this final state is YES, we say that $\mathcal{M}$ *accepts* the input string x (we denote this fact by $\mathcal{M}(x) = \textsc{yes}$), and if the final state is NO, we say that $\mathcal{M}$ *rejects* x (in this case we write $\mathcal{M}(x) = \textsc{no}$). Otherwise, HALT was reached, and since the computation has gone on for finitely many steps, the string on the $\mathcal{M}$'s tape at the time of halting consists of a $\rhd$, followed by a finite string $y \in \Sigma^*$, followed by a symbol from $\Gamma - \Sigma$, followed by an infinite sequence of symbols from Γ. We consider string y to be the *output* of the computation, and write $\mathcal{M}(x) = y$.

The transition function δ defines the DTM's behavior and can be viewed as a formal definition of the notion of a (deterministic) algorithm. Thus, every deterministic Turing machine describes an algorithm, and every algorithm represents a deterministic Turing machine. We will use these two notions interchangeably.

The following definitions clarify what does it mean to decide a language or to compute a function.

Definition 2.23. *Let $\mathcal{A}$ be an algorithm represented by a deterministic Turing machine $\mathcal{M} = (Q, \Gamma, \Sigma, \delta, q_0)$ and let $L \subseteq \Sigma^*$ be a language. We say that $\mathcal{A}$ decides language L if, for each string $x \in \Sigma^*$, it holds that $\mathcal{M}(x) = \textsc{yes}$, if $x \in L$, and $\mathcal{M}(x) = \textsc{no}$, if $x \notin L$. For some function $T \colon \mathbb{N} \mapsto \mathbb{N}$ we say that $\mathcal{A}$ decides language L in time $T(n)$, if $\mathcal{A}$ decides L and when it is provided with an input string $x \in \Sigma^*$, it stops after at most $T(|x|)$ steps.*

Definition 2.24. *Let $\mathcal{A}$ be an algorithm represented by a deterministic Turing machine $\mathcal{M} = (Q, \Gamma, \Sigma, \delta, q_0)$ and let $f \colon \Sigma^* \mapsto \Sigma^*$ be a function. We say that $\mathcal{A}$ computes function f if, for each string $x \in \Sigma^*$, it holds that $\mathcal{M}(x) = y$ if and only if $f(x) = y$. For some* *function $T\colon \mathbb{N} \mapsto \mathbb{N}$ we say that $\mathcal{A}$ computes function f in time $T(n)$, if $\mathcal{A}$ computes f and when it is provided with an input string $x \in \Sigma^*$, it stops after at most $T(|x|)$ steps.*

Function $T(n)$ from the definitions above is called a *time complexity* of algorithm $\mathcal{A}$ (or machine $\mathcal{M}$). In most cases we are not interested in the exact form of the time complexity. In other words, if there is a function $T'(n)$ and a positive number a for which it holds

$$\lim_{n\to\infty} \frac{T(n)}{T'(n)} = a,$$

we can use $T'(n)$ for algorithm's time complexity instead of $T(n)$. For example, if $T(n)$ is a polynomial of degree k, we can consider only the leading term of $T(n)$, since the lower-order terms are relatively insignificant for large n. We also ignore the leading term's constant coefficient, since constant factors are less significant than the rate of growth in determining computational efficiency for large inputs. In effect we use $T'(n) = n^k$ instead of $T(n)$. Therefore, when studying the complexity of algorithms we use the notations O, Ω and Θ that we described in the previous section.

We are particularly interested in the case when the time complexity of an algorithm is a polynomial of the input string's length. These algorithms form the complexity class P, whose formal definition is as follows:

$$\begin{aligned} \mathrm{P} = \{\, L \subseteq \{0,1\}^* : & \text{ there exists an algorithm } \mathcal{A} \\ & \text{that decides } L \text{ in polynomial time} \,\}. \end{aligned}$$

Generally, we think of problems that are solvable by polynomial-time algorithms (i.e., languages from P) as being tractable, or easy, and problems that require superpolynomial time as being intractable, or hard. Although it is reasonable to regard a problem of time boundary $T(n) = O(n^{100})$ as intractable, there are very few practical problems that require time on the order of such a high-degree polynomial. Experience has shown that once a polynomial-time algorithm for a problem is discovered, it is likely that an algorithm with a much better running time will soon be discovered.

In this thesis we describe algorithms not by giving the whole configuration of Turing machines that implement them, but rather we write them in a *pseudocode* designed to present each algorithm clearly and succinctly and to be readable by anyone familiar with computer programming. Although we use the natural English language in many situations when writing in pseudocode, its translation into most modern structured programming languages is a fairly straightforward task.

Example 2.8. Let us consider the following algorithm written in pseudocode:

```
IS-PRIME($n$)
1  if $n = 0$ or $n = 1$
2     then return "no"
3  for $k = 2$ to $\lfloor\sqrt{n}\rfloor$
4        do if $n$ is evenly divisible by $k$
5              then return "no"
6  return "yes"
```

The reader can easily check the algorithm solves problem Π'' (or decides language PRIMES) from Example 2.7. Since n is a natural number encoded in binary, it consists of $m = \lfloor \log_2 n \rfloor + 1$ bits, and $\lfloor \sqrt{n} \rfloor$ encoded in binary consists of $\lfloor \log_2 \lfloor \sqrt{n} \rfloor \rfloor + 1 < (\log_2 n)/2 = O(m)$ bits. Hence, the test in line 4 is executed $O(2^m)$ times, and the time complexity of this algorithm is exponential.

Of course, there are other algorithms which solve Π'' and that differ in computational complexity. In particular, Agrawal, Kayal, and Saxena [AKS04] invented polynomial-time primality test, therefore we can write PRIMES $\in$ P. □

Let us now move on to the *non-deterministic* version of Turing machines (NDTM).

Definition 2.25. *A non-deterministic Turing machine is a 5-tuple $\mathcal{N} = (Q, \Gamma, \Sigma, \Delta, q_0)$, where Q, Γ, Σ and q_0 have the same meaning as in the definition of deterministic Turing machine, and $\Delta \subseteq (Q \times \Gamma) \times (\Gamma \times \{\leftarrow, \rightarrow\} \times (Q \cup \{\textsc{halt}, \textsc{yes}, \textsc{no}\}))$ is a transition relation. In addition, we assume that $Q \cap \Gamma = \emptyset$ and $\leftarrow, \rightarrow, \textsc{halt}, \textsc{yes}, \textsc{no} \notin \Gamma$, and for each $p, q \in Q$, if $((p, \triangleright), (s, d, q)) \in \Delta$ then $s = \triangleright$ and $d = \rightarrow$.*

The main difference between DTM and NDTM is that the latter is able to choose the way the computation will go, since given the current state and the symbol in the current cell, there can be more than one possibility specified what the next step may be. An NDTM *accepts* an input string if there is some sequence of non-deterministic choices that results in YES state. An NDTM *rejects* an input string only if no sequence of choices can lead to acceptance. This "asymmetry" in the way "yes" and "no" instances are treated in non-deterministic machines is the reason why such machines are so powerful.

Every NDTM constitutes a definition of a *non-deterministic algorithm* similarly as it is for DTMs and deterministic algorithms. Definition 2.23 remains true if we redefine $\mathcal{A}$ to be a non-deterministic algorithm represented by NDTM $\mathcal{N} = (Q, \Gamma, \Sigma, \Delta, q_0)$. We consider the analogous class of problems solvable (i.e., the class of languages decided) in polynomial-time on NDTM. Its definition is as follows:

$$\begin{aligned} \mathrm{NP} = \{\, L \subseteq \{0,1\}^* : {} & \text{there exists a non-deterministic algorithm } \mathcal{A} \\ & \text{that decides } L \text{ in polynomial time} \,\}. \end{aligned}$$

Of course, every transition function from the definition of DTM is a transition relation from the definition of NDTM. Thus, every deterministic Turing machine is a non-deterministic Turing machine. From this observation we have that P $\subseteq$ NP. If the opposite inclusion is true we would have that P = NP, but this question, so-called "P $\stackrel{?}{=}$ NP problem", is one of the deepest open research problems in theoretical computer science. For the discussion about what the consequences of any solution to this problem would be, and the recent status of its exploration, see the paper of Fortnow [For09].

We will now formalize the notion of reducing one problem to another. Intuitively, a problem Π can be reduced to another problem Π' if any instance of Π can be "easily rephrased" as an instance of Π', the solution to which provides a solution to the instance of Π.

Definition 2.26. *Let $\Sigma = \{0,1\}$ be an alphabet and let L_1 and L_2 be languages over Σ. Language L_1 reduces in polynomial-time to L_2, denoted $L_1 \leq_{\mathrm{P}} L_2$, if there exists a polynomial-time computable function $f\colon \Sigma^* \mapsto \Sigma^*$ such that, for all $x \in \Sigma^*$, $x \in L_1$ if and only if $f(x) \in L_2$.*

Polynomial-time reductions give us a powerful tool for proving that various problems belong to P. For example, let $L_2 \in \mathrm{P}$. If $L_1 \leq_{\mathrm{P}} L_2$, it holds that $L_1 \in \mathrm{P}$, because we can transform L_1 to L_2 using polynomial-time reduction and solve L_2 in polynomial-time. In other words polynomial-time reductions provide a formal means for showing that one problem is at least as hard as another, to within a polynomial-time factor.

We can now define a set of NP-complete languages which are the hardest problems in NP.

Definition 2.27. *A language $L \subseteq \{0,1\}^*$ is* NP*-complete, if*

1. $L \in \mathrm{NP}$, *and*
2. $L' \leq_{\mathrm{P}} L$ *for every* $L' \in \mathrm{NP}$.

If a language L satisfies property 2, but not necessarily property 1, we say that L is NP*-hard.*

Once we find a polynomial-time algorithm for any NP-complete problem, we will automatically receive polynomial-time algorithms for each problem in this class, and it will be a positive answer to the $\mathrm{P} \overset{?}{=} \mathrm{NP}$ problem. For decades these problems were unsuccessfully attacked by many researchers, it is therefore believed that fast algorithms that solve NP-complete problems do not exist.

We will need a notion of *oracle machine*. It can be visualized as a deterministic Turing machine with a black box, called an oracle, which is thought of as an entity capable of answering some collection of questions. An oracle machine is able to solve decision problem Π in a single operation, no matter how hard Π is (i.e., Π can be of any complexity class).

Let us define a complexity class which is not as popular as P or NP, but we need it to precisely describe some properties of elections. This class is denoted by Θ_2^p and it consists of problems which can be solved in polynomial time on a deterministic Turing machine with an oracle for a problem in NP, by $O(\log_2 n)$ queries to the oracle (where n is the size of input). A problem Π is Θ_2^p*-complete*, if it belongs to Θ_2^p, and every problem in Θ_2^p reduces to Π in polynomial-time.

Let us now move on to function problems. We start by giving a definition of the complexity class of "easy" function problems which is analogous to class P:

$$\mathrm{FP} = \{\, f\colon \{0,1\}^* \mapsto \{0,1\}^* : \text{there exists an algorithm } \mathcal{A} \text{ that computes } f \text{ in polynomial time} \,\}.$$

Let Π be some decision problem where, for each instance I, we ask if there exists some mathematical object satisfying a given condition. The *counting variant* of Π is denoted $\#\Pi$. It is an example of a function problem, in which we ask how many given mathematical objects exist. For example, consider the following definition.

Definition 2.28. *An instance of* X3C *is a pair* $(B, \mathcal{S})$*, where* $B = \{b_1, \dots, b_{3k}\}$ *and* $\mathcal{S} = \{S_1, \dots, S_n\}$ *is a family of 3-element subsets of* B*. In* X3C *we ask if it is possible to find exactly* k *sets is* $\mathcal{S}$ *whose union is exactly* B*. In* #X3C *we ask how many* k*-element subsets of* $\mathcal{S}$ *have* B *as their union.*

The full name of X3C problem is *exact 3-set cover*. It is known to be NP-complete (see Garey and Johnson [GJ90]).

The complexity class #P is the set of such counting problems #Π for which there is a polynomial-time NDTM that accepts on as many paths as many solutions there are for Π. Before specifying what does it mean for a counting problem to be #P-complete, let us define the notion of reduction between counting problems. It is similar but more subtle than the notion of reduction between decision problems.

Definition 2.29. *Let* $\#\Pi_1$ *and* $\#\Pi_2$ *be two counting problems. We say that* $\#\Pi_1$ *parsimoniously reduces to* $\#\Pi_2$*, denoted* $\#\Pi_1 \leq_{\mathrm{par}} \#\Pi_2$*, if there exists a function* $f \in \mathrm{FP}$ *such that for each instance* I *of* $\#\Pi_1$ *the following two conditions hold:*

1. $f(I)$ *is an instance of* $\#\Pi_2$*, and*
2. I *has exactly as many solutions as* $f(I)$*.*

The definition above is a basis of the notion of #P-completeness.

Definition 2.30. *We say that a counting problem* $\#\Pi$ *is* #P*-parsimonious-complete, if*

1. $\#\Pi \in \#\mathrm{P}$*, and*
2. $\#\Pi' \leq_{\mathrm{par}} \#\Pi$ *for every* $\#\Pi' \in \#\mathrm{P}$*.*

If $\#\Pi$ *satisfies property 2, but not necessarily property 1, we say that* $\#\Pi$ *is* #P*-parsimonious-hard.*

Many counting problems are #P-parsimonious-complete if their decision variants are NP-complete. #X3C is an example of a problem with such property. However, it is not a rule, as parsimonious reductions are quite restrictive and the class of #P-parsimonious-complete problems is not as wide as the class of NP-complete problems. Different authors sometimes use different reduction types to define #P-completeness and #P-hardness. For example, Zankó [Zan91] used many-one reductions, Krentel [Kre88] used metric reductions, and Valiant [Val79] used Turing reductions. These definitions are given below.

Definition 2.31 (Zankó)**.** *Let* $\#\Pi_1$ *and* $\#\Pi_2$ *be two counting problems. We say that* $\#\Pi_1$ *many-one reduces to* $\#\Pi_2$*, denoted* $\#\Pi_1 \leq_{\mathrm{m}} \#\Pi_2$*, if there exist functions* f*,* $g \in \mathrm{FP}$ *such that for each instance* I *of* $\#\Pi_1$ *the following two conditions hold:*

1. $f(I)$ *is an instance of* $\#\Pi_2$*, and*
2. *if* s_I *is the number of solutions of* $f(I)$*,* I *has exactly* $g(s_I)$ *solutions.*

Definition 2.32 (Krentel)**.** *Let* $\#\Pi_1$ *and* $\#\Pi_2$ *be two counting problems. We say that* $\#\Pi_1$ *metrically reduces to* $\#\Pi_2$*, denoted* $\#\Pi_1 \leq_{1-T} \#\Pi_2$*, if there exist functions* f*,* $g \in \mathrm{FP}$ *such that for each instance* I *of* $\#\Pi_1$ *the following two conditions hold:*

1. *$f(I)$ is an instance of $\#\Pi_2$, and*
2. *if s_I is the number of solutions of $f(I)$, I has exactly $g(I, s_I)$ solutions.*

Definition 2.33 (Valiant). *Let $\#\Pi_1$ and $\#\Pi_2$ be two counting problems. We say that $\#\Pi_1$ is Turing reducible to $\#\Pi_2$, denoted $\#\Pi_1 \leq_T \#\Pi_2$, if there is an oracle machine that computes $\#\Pi_1$ with $\#\Pi_2$ as the oracle.*

The meaning of #P-completeness and #P-hardness in the context of each type of reduction is similar to that from Definition 2.30 in which the property 2 has been changed appropriately. Parsimonious reduction is stronger than many-one reduction, which is stronger than metric reduction, which in turn is stronger than Turing reduction. In other words, if $\#\Pi_1 \leq_{\mathrm{par}} \#\Pi_2$ then $\#\Pi_1 \leq_{\mathrm{m}} \#\Pi_2$, and so on. It means that for example the class of #P-parsimonious-complete problems lies within class of #P-many-one-complete problems and, provided that FP $\neq$ #P, these inclusions are strict (see [FO10] for an argument).

Chapter 3

Elections and Election-Related Problems

In order to study elections from the point of view of computational complexity theory, we need to transform the intuitive sense of what elections and voting systems are into a precise mathematical model. The current chapter includes formal definitions of these notions, and specifications of control problems we will study under several voting systems. We also make an overview of the most common voting rules along with the demonstration of how they work for some specified collections of candidates and voters. We point the reader to [ASS02] for general reference.

3.1. Elections and Voting Systems

We begin by formalizing the central notion of this paper—an election.

Definition 3.1. *An election E is an ordered pair (C, V) where C is a finite non-empty set of candidates and V is a finite non-empty multiset of voters, where each voter is represented by a total order over C.*

In this work we use m to denote the number of candidates in C and n to denote the number of voters in V, and assume that $C = \{c_0, c_1, \ldots, c_{m-1}\}$. Voters vote on the candidates by setting their preference lists, i.e., by putting the candidates in order from the most preferred one to the most despised one. Thus, $V = \{\succ_0, \succ_1, \ldots, \succ_{n-1}\}$ is a multiset over the set of all total orders over C. In most cases we study, such definition of election is sufficient, although in many situations we do not have the full information about voters' preferences. We may have only partial knowledge and do not know what the rest of the voter's preference is. Thus, every element of V can be treated as a partial order over C and in this case we say $E = (C, V)$ is a *partial election.*

Example 3.1. Let us consider an election $E = (C, V)$, where:

$$\begin{aligned}
&C = \{c_0, c_1, c_2\},\\
&V = \{c_1 \succ_0 c_2 \succ_0 c_0,\ c_2 \succ_1 c_1 \succ_1 c_0,\ c_0 \succ_2 c_2 \succ_2 c_1,\ c_1 \succ_3 c_2 \succ_3 c_0\}.
\end{aligned}$$

We have $m = 3$ candidates and $n = 4$ voters, where the first and the fourth voter provide the same ranking: They prefer candidate c_1 over c_2, and c_2 over c_0. $\square$

Once we have an election, we need some formal rule—or protocol—for determining the winner(s) of the election. We call this rule a *voting system.* Consider the following definition.

Definition 3.2. *A voting system $\mathcal{E}$ is defined to be a function mapping an election $E = (C, V)$ to a set $W \subseteq C$ of candidates who win the election E.*

We also call $\mathcal{E}$ a voting rule, a voting protocol, or a voting scheme. The members of the set W from the definition above are called $\mathcal{E}$-winners—or simply winners—of election E when a voting system is known from a context. The set W may be empty if there are no winners of the election, may be a singleton—we will then say that the election has a *unique winner*—or may have more than one element.

There are very many voting systems. We, in particular, study some of the so-called *scoring protocols* and some of *Condorcet-consistent rules.*

Scoring protocols are an important class of voting systems. Given m candidates, each scoring protocol is defined by a *scoring vector* $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_m)$ satisfying $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_m \geq 0$. When a candidate is placed on the ith position in a voter's preference order, he receives α_i points. After summing up the scores granted by each voter, the candidate(s) with the highest score are the winners of the election. Given a scoring system, we write $score_E(c_j)$ to mean the score of candidate c_j in election E. The most common scoring systems are:

1. *plurality voting*, with the scoring vector $\boldsymbol{\alpha}_{Plurality} = (1, \overbrace{0, \ldots, 0}^{m-1})$,
2. *veto voting*, with the scoring vector $\boldsymbol{\alpha}_{Veto} = (\overbrace{1, \ldots, 1}^{m-1}, 0)$, and
3. *Borda count*, with the scoring vector $\boldsymbol{\alpha}_{Borda} = (m-1, m-2, \ldots, 0)$.

Let us consider once again election E from Example 3.1, and determine which candidates win under the three voting systems defined above. In plurality rule c_1 receives 2 points, while other candidates get 1 point each, so clearly c_1 is the unique winner of E. Formally, we write $\mathit{Plurality}(E) = \{c_1\}$. Under veto rule, c_2 receives 4 points, c_1 receives 3 points, and c_0 receives only 1 point, so $\mathit{Veto}(E) = \{c_2\}$. In Borda voting we have that both c_1 and c_2 get 5 points, while c_0 gets only 2 points, thus $\mathit{Borda}(E) = \{c_1, c_2\}$. This example shows that our choice of the voting rule is very important, because under different rules we may get different sets of winners. The other thing to notice is that the more voters rank some candidate low, the less chance this candidate has to be a winner. This is a direct consequence of the fact that the scoring vector is a non-increasing sequence.

Another type of voting system is *approval voting.* Its structure is slightly different from the other systems, as we do not treat voters' preference orders as total or partial orders but as subsets of C. Each voter $v_i \in V$ provides a subset $A_i \subseteq C$ of candidates that he approves of. A candidate $c \in C$ is a winner, when he appears in the largest number of such subsets.

Example 3.2. Let $E = (C, V)$, where:

$$C = \{c_0, c_1, c_2\},$$
$$V = \{\underbrace{\{c_0, c_1, c_2\}}_{A_0}, \underbrace{\{c_0, c_1, c_2\}}_{A_1}, \underbrace{\{c_0\}}_{A_2}, \underbrace{\{c_0, c_1, c_2\}}_{A_3}\}.$$

Voters' preferences are similar to those in Example 3.1, but they do not provide the order of candidates in each vote. Each voter specifies the set of candidates that he approves of. We can see that every voter but the third approves of all the three candidates, and the remaining one approves only of c_0. With this configuration, c_0 gets 4 points, and other candidates get 3 points each. Thus $\mathit{Approval}(E) = \{c_0\}$. This example shows that it is possible to secure victory of a candidate even though it might seem unlikely under other voting schemes, since c_0 is not a winner of the similar election from Example 3.1 under every other voting scheme considered in this section. $\square$

There is also a modified version of this election system, called *k-approval voting* (for integer k such that $1 \leq k \leq m$) in which we demand that every voter chooses exactly k candidates, i.e., $|A_i| = k$, for each integer i, $1 \leq i \leq n$. k-approval can also be viewed as a scoring protocol with scoring vector $\boldsymbol{\alpha}_{k\text{-}Approval} = (\underbrace{1, \dots, 1}_{k}, \underbrace{0, \dots, 0}_{m-k})$. Note that the case $k = m$ is trivial, as the whole C is the set of winners, if $k = 1$, this system is equivalent to plurality voting and if $k = m - 1$ then we have veto. When $1 \leq k \leq m - 1$, $(m - k)$-approval is often called *k-veto voting*.

The second class of voting rules that we study here are those that consider results of pairwise comparisons of candidates. An important and well studied rule is *Condorcet voting* in which $c \in C$ is a winner if and only if for each $c' \in C - \{c\}$, $c \succ_i c'$ for more than $n/2$ of preference orders $\succ_i$ from V. There can be at most one winner under Condorcet's rule and he is called the *Condorcet winner*. We define $N_E(c_j, c_k)$ to be the number of preference orders $\succ_i$ from V, such that $c_j \succ_i c_k$. Thus, c is a Condorcet winner exactly, if $N_E(c, c') > N_E(c', c)$ for each $c' \in C - \{c\}$.

In Example 3.1 there is no Condorcet winner as c_1 wins with c_2 in 2 votes, c_2 wins over c_1 also in 2 votes, and c_0 wins over both c_1 and c_2 only in 1 vote. Therefore, we can write $\mathit{Condorcet}(E) = \emptyset$.

The last voting system we consider in this work, called *maximin voting*, is based on scoring while making use of some of the ideas from Condorcet rule. In maximin system, the score of candidate $c \in C$ is defined as $\min_{c' \in C - \{c\}} N_E(c, c')$. The score of candidate c_0 from Example 3.1 is 1, and the score of c_1 and c_2 is 2, thus $\mathit{Maximin}(E) = \{c_1, c_2\}$.

3.2. Control Problems in Elections and Their Counting Variants

Control is an attack widely used in real-world elections. Although many types of elections are performed secretly, i.e., nobody but the voter knows his own preferences, there are situations where some actor (or a group of actors) has partial or complete knowledge of all the votes. In such situations this actor may seek to determine the way how the parameters of the election can be changed in order to achieve the desired outcome. Political parties often keep track of the statistics and pre-election polls and develop advertisements directed at particular groups of people in order to convince them to vote. Another example of control attack can be observed in song of the month contests. A radio station may promote new songs by adding them to the list and increasing their chances for gaining popularity, hence increasing the producer income.

Given an election, we ask if it is possible to modify it in some way in order to make the distinguished candidate a winner (constructive control problem), or to prevent this candidate from being a winner (destructive control problem). The types of such modifications will be precisely described later in this section, but in short, we can add or delete some candidates or voters. Partitioning candidates or voters into a 2-round structure is also studied in the literature along with two different tie-breaking rules, but in this work we do not discuss these types of control. The family of control problems has been formulated and studied by Bartholdi, Tovey, and Trick in [BTT89b] and by Hemaspaandra, Hemaspaandra, and Rothe in [HHR07]. These papers show that some of the modifications are computationally easy to perform in order to solve a problem, while the others are computationally hard, or even never lead to a solution.

In this work we focus on the counting variants of control problems rather than on the decision ones. Therefore, we are interested not only in determining if a solution to the problem exists, but also in the existence of fast algorithms that count all the solutions to the given problem.

We now formally define all problems we study in this paper. Every problem we consider has received a name which is made of the name of the election system $\mathcal{E}$; a short prefix defining the type, such as 'AV' for adding voters; one of '$_{\text{C}}$' or '$_{\text{D}}$' subscripts denoting that the problem is either the constructive case or the destructive case; and a description part which is 'Control' in each of them. To distinguish them from the decision variants, we put a '#' sign after election system part. If we study the problem given some election system, say approval voting, we write '*Approval*' in the description part instead of $\mathcal{E}$.

Counting Variant of Control by Adding Candidates

Name: $\mathcal{E}$-#AC$_{\text{C}}$-Control

Given: An election $E = (C, V)$, a set C' of additional candidates with $C \cap C' = \emptyset$, a distinguished candidate $c \in C$, and a positive integer $k \leq |C'|$. Each voter in V provides preferences over $C \cup C'$.

Question: How many sets $B \subseteq C'$ with $|B| \leq k$ are there, such that c is the unique $\mathcal{E}$-winner of election $E' = (C \cup B, V)$?

Counting Variant of Control by Deleting Candidates

Name: $\mathcal{E}$-#DC$_{\text{C}}$-Control

Given: An election $E = (C, V)$, a distinguished candidate $c \in C$, and a positive integer $k \leq m$.

Question: How many sets $B \subseteq C$ with $|B| \leq k$ are there, such that c is the unique $\mathcal{E}$-winner of election $E' = (C - B, V)$?

Counting Variant of Control by Adding Voters
Name: $\mathcal{E}$-#AV$_\mathrm{C}$-Control
Given: An election $E = (C, V)$, a multiset V' of additional voters, $V \cap V' = \emptyset$, with preferences over C, a distinguished candidate $c \in C$, and a positive integer $k \leq |V'|$.
Question: How many multisets $U \subseteq V'$ with $|U| \leq k$ are there, such that c is the unique $\mathcal{E}$-winner of election $E' = (C, V \uplus U)$?

Counting Variant of Control by Deleting Voters
Name: $\mathcal{E}$-#DV$_\mathrm{C}$-Control
Given: An election $E = (C, V)$, a distinguished candidate $c \in C$, and a positive integer $k \leq n$.
Question: How many multisets $U \subseteq V$ with $|U| \leq k$ are there, such that c is the unique $\mathcal{E}$-winner of election $E' = (C, V - U)$?

The instances of each problem above are tuples constructed from the elements listed in the "Given" parts. For example, in $\mathcal{E}$-#AV$_\mathrm{C}$-Control each instance is of the form (E, V', c, k). The set of solutions in every case is of course the set of naturals, as in all counting problems.

The destructive versions of control problems differ only in the "Question" part, which demands that c is *not* the unique $\mathcal{E}$-winner of the appropriately defined election E'. Moreover, in $\mathcal{E}$-#DC$_\mathrm{D}$-Control we prevent deleting c (also assuming that $k < m$), since otherwise any voting system could have been trivially controlled.

When regarding decision variants of control problems we say a voting system is *immune* to control if it is not possible for the chairman to change a distinguished candidate from a non-winner to a unique winner (or inversely in the destructive variant) by manipulating the set of candidates or the multiset of voters. Otherwise, we regard two distinct cases—if for an election system it is hard (i.e., NP-complete) to find a way to control the election, the system is referred to as *resistant*, and if this is always computationally easy (i.e., the problem is in P), the system is referred to as *vulnerable*. Of course, in the case of immunity the counting variant of control problem is trivially in FP, as we always return 0; the other cases require a proof of either being in FP or being #P-complete. As we have seen in the previous chapter, if some problem is NP-complete it does not automatically mean that its counting variant is #P-complete (however, it is typically the case).

3.3. Winner Prediction Model

We will now describe how winner prediction relates to control problems. Let us say we are given a voting rule $\mathcal{E}$, a set C of m candidates, and a multiset V of n voters (for each voter we have perfect knowledge as to how he would vote). We consider the situation where each candidate from a set D, $D \subseteq C$, may or may not participate in the election, and similarly, where each voter from a multiset W, $W \subseteq V$, may or may not actually vote. For such uncertain election $E = (C, V)$ we ask for the probability that a designated candidate $c \in C$ is the unique winner under voting system $\mathcal{E}$.

Let us study an example scenario, showing the connection between problem $\mathcal{E}$-#AV$_C$-Control and the appropriate type of winner prediction; the reader can imagine analogous settings for the remaining types of control. Let us assume that set C of candidates participating in the election is fixed (for example, because the election rules force all candidates to register well in advance). We know that some multiset V of voters will certainly vote (for example, because they have already voted and this information is public). The collection of voters who have not decided to vote yet is W. From some source (e.g., from prior experience) we have some probability distribution P on the number of voters from W that will participate in the election (from our perspective, each equal-sized subcollection of voters from W is equally likely; different-sized collections may, of course, have different probabilities of participating in the election).

In other words, for each i, $0 \le i \le |W|$, let $P(i)$ be the probability that i voters from W join the election (and assume that we have an easy way of computing this value) and let $Q(i)$ be the probability that a designated candidate c wins under the condition that exactly i voters from W participate (assuming that each i-element subcollection of W is equally likely). Then, the probability that c wins is simply given by:

$$P(0)Q(0) + P(1)Q(1) + \cdots + P(|W|)Q(|W|).$$

To compute $Q(i)$, we have to compute for how many collections W of size exactly i, candidate c wins, and divide it by $\binom{|W|}{i}$. To compute for how many sets of size exactly i candidate c wins, we solve the corresponding $\mathcal{E}$-#AV$_C$-Control problem for adding at most i voters from W, then for adding at most $i - 1$ voters from W, and then we subtract the results.

In our setting we are given two probability distributions, P_C and P_V, defined on the set of all subsets of C and, respectively, on the set of every subcollection of V. We consider two possible scenarios:

1. The set of candidates is fixed (i.e., $P_C(C) = 1$), but for each multiset of voters V', $V' \subseteq V$, we have probability $P_V(V')$ that exactly the voters from V' show up for the vote.
2. The multiset of voters is fixed (i.e., $P_V(V) = 1$), but for each set of candidates C', $C' \subseteq C$, we have probability $P_C(C')$ that exactly the candidates from C' participate in the election.

Naturally, out task would very quickly become computationally prohibitive if we did not assume anything about P_C and P_V. Of course, first of all we need to assume that both probabilities, P_C and P_V, are polynomial-time computable. We also would like to assume that functions P_C and P_V depend only on the cardinality of their arguments. More formally, if C_1, $C_2 \subseteq C$ and $|C_1| = |C_2|$, then $P_C(C_1) = P_C(C_2)$, and analogously for P_V. In other words, we consider probability distribution regarding the number of candidates (the number of voters) participating in the election, but each same-cardinality subset is equally likely.

However, we will focus on some special situations that will allow us to easily transform our setting to counting variants of control problems. In order to do this, we need to slightly weaken the assumption regarding the dependency only on the sizes of candidate/voter sets. Often, we may have additional knowledge regarding the nature of possible changes in the candidate/voter set. For example, the rules may be such that after a given point of time candidates can withdraw from the election but no new candidates can register. Similarly, we may know that some votes have already been cast and cannot be withdrawn. Thus, we refine our model to be the following: We start with some candidates and voters already in the election, and we ask for the probability that a given candidate wins assuming that some random number of voters/candidates is added/deleted.

More formally, we start with an election $E = (C, V)$ and we consider the following situations:

1. The set of candidates C is fixed, and each voter from V will vote, but there is a multiset of unregistered voters V', $V' \cap V = \emptyset$, that also may show up for the vote.
2. The set of candidates C is fixed, but some of the voters from V may not show up for the vote.
3. The multiset of voters V is fixed, and each candidate from C will participate in the election, but there is a set of unregistered candidates C', $C' \cap C = \emptyset$, that also may participate.
4. The multiset of voters V is fixed, but some of the candidates from C may not participate in the election.

In effect, after applying one of these scenarios to E we obtain an election E'. For example, in the second case above we consider $E' = (C, V - U)$, where $U \subseteq V$. We also use a fixed upper bound for the number of candidates/voters that can be added/deleted in election E.

One can easily verify that the prediction of the winner in the presented setting reduces to the counting variants of control problems which have been defined in the previous section.

In the following chapters we study counting variant of control problems under several voting systems. If for a voting system $\mathcal{E}$ a particular control problem is computationally easy, so is the corresponding type of winner prediction problem under $\mathcal{E}$.

Chapter 4

Control Problems in Plurality Voting

Under plurality voting, counting variants of both control by adding candidates and control by deleting candidates are #P-complete, as we show it in the following theorems. However, polynomial-time algorithms exist for the cases where we modify the collection of voters.

4.1. Adding and Deleting Candidates

The first problem we consider under plurality voting is constructive control by adding candidates. Below we show that this problem is #P-parsimonious-complete (the proof approach is inspired by a related result in [FHH11a]).

Theorem 4.1. *Plurality-*#AC$_{\mathrm{C}}$-CONTROL *is* #P*-parsimonious-complete.*

Proof. In the first step we need to show that *Plurality-*#AC$_{\mathrm{C}}$-CONTROL $\in$ #P. From the definition of class #P, we need give polynomial-time NDTM that accepts on as many computational paths as many solutions there are for our problem. We can imagine a non-deterministic algorithm that, given an instance $I = (E, C', c, k)$, it simply checks all possible subsets of C' of sizes at most k, if any of them consists of exactly those new candidates whose addition to C makes c a unique winner. This requires only polynomially many steps because it can non-deterministically choose the next candidate from C' in each step.

To prove that *Plurality-*#AC$_{\mathrm{C}}$-CONTROL is #P-parsimonious-hard, we show that there is a parsimonious reduction from the #X3C problem (see Definition 2.28).

Suppose we are given an instance of #X3C, $(B, \mathcal{S})$, where $B = \{b_1, \ldots, b_{3k}\}$ and $\mathcal{S} = \{S_1, \ldots, S_r\}$ with $r \geq 4$ and $k \geq 2$. For each $b_i \in B$, we set ℓ_i to be the number of sets in $\mathcal{S}$ that contain b_i.

We construct an election $E = (C, V)$, where $C = B \cup \{c, w\}$ is the set of registered candidates, and V is a multiset of voters. We also have a set $A = \{a_1, \ldots, a_r\}$ of spoiler (unregistered) candidates. Each candidate a_i in A corresponds to a set S_i in $\mathcal{S}$. Collection V contains groups of voters with preferences over $C \cup A$ described below.

For each vote we specify up to two top-ranked candidates; the remaining candidates are ranked in arbitrary order in the '$\cdots$' part.

1. For each set $S_j \in \mathcal{S}$, for each $b_i \in S_j$, we have $2k$ votes $a_j \succ b_i \succ \cdots$.
2. For each set $S_j \in \mathcal{S}$, we have 1 vote $a_j \succ c \succ \cdots$.
3. We have $2rk + k - r + 1$ votes $c \succ \cdots$.
4. We have $2rk$ votes $w \succ \cdots$.
5. For each $b_i \in B$, we have $2rk + 2k - 2k\ell_i$ votes $b_i \succ \cdots$.

Thus in election E the scores of candidates are as follows:

1. $score_E(c) = 2rk + k + 1$,
2. $score_E(w) = 2rk$, and
3. for each $b_i \in B$, $score_E(b_i) = 2rk + 2k$.

That is, the winners of plurality election E are exactly the candidates in B. We claim that every set A', $A' \subseteq A$, such that $|A'| \leq k$ and that c is a unique winner of plurality election $E' = (C \cup A', V)$ corresponds one-to-one to a family $\mathcal{S}'$, $\mathcal{S}' \subseteq \mathcal{S}$, such that $|\mathcal{S}'| = k$ and $\bigcup \mathcal{S}' = B$ (i.e., $\mathcal{S}'$ is an exact set cover of B).

Let us fix set A'. It corresponds to a family $\mathcal{S}'$ of 3-sets from $\mathcal{S}$ such that for each integer j, $1 \leq j \leq r$, $\mathcal{S}'$ contains set S_j if and only if A' contains a_j. It is easy to see that in election $E' = (C \cup A', V)$ the scores of candidates are as follows:

1. $score_{E'}(c) = 2rk + k + 1 - |A'|$,
2. $score_{E'}(w) = 2rk$,
3. for each $b_i \in B$, $score_{E'}(b_i) = 2rk + 2k - 2k\,|\{a_j \in A' : b_i \in S_j\}|$, and
4. for each $a_j \in A'$, $score_{E'}(a_j) = 6k + 1$.

Assume that c is a unique winner of election E'. Then c's score is the highest among the scores of other candidates. Inequality $score_{E'}(c) > score_{E'}(a_j)$ holds for each $a_j \in A'$ under assumption that $r \geq 4$. In order to have $score_{E'}(c) > score_{E'}(w)$, we need to assume that $|A'| \leq k$, and thus $score_{E'}(c) \geq 2rk + 1$. For each $b_i \in B$ there is $score_{E'}(c) > score_{E'}(b_i)$, which holds if $score_{E'}(b_i) \leq 2rk$, so we have that $|\{a_j \in A' : b_i \in S_j\}| \geq 1$. It means that each $b_i \in B$ belongs to at least one set S_j. The score of each of the $3k$ candidates in B has to be decreased and each $a_j \in A'$ corresponds to decreasing the score of exactly 3 candidates in B, because every S_j contains exactly 3 distinct elements of B. Thus, A' corresponds to an exact set cover of B.

On the other hand, let us assume A' corresponds to an exact set cover of B. In such a case $|A'| = k$ because in an exact set cover of $3k$-element set we must have exactly k 3-element sets. Thus, $score_{E'}(c) = 2rk + 1$. Since each $a_j \in A'$ corresponds to a set $S_j \in \mathcal{S}$ that contains 3 distinct members of B, for each $b_i \in B$ we have that $|\{a_j \in A' : b_i \in S_j\}| = 1$ and therefore $score_{E'}(b_i) = 2rk < score_{E'}(c)$. For each $a_j \in A'$ it holds $score_{E'}(a_j) = 6k + 1 < 2rk + 1 = score_{E'}(c)$ (since we assumed that $r \geq 4$). As we can see, c has the greatest score, so it is a unique winner of election E'. $\square$

In our further proofs in which we show reductions from #X3C problem, we will not be as precise and we will be describing things less explicitly. In particular, we assume that the reader can easily verify that a problem belongs to class #P based on the method in the proof above.

Thanks to the proved fact, we can easily show #P-completeness (in the sense of Krentel) of the destructive control by adding candidates. In our proof the answer to the problem depends on its instance as well as on the number of solutions in the constructive case. Although we do not show the problem's possible #P-parsimonious-completeness, we do not claim it does not hold.

Theorem 4.2. *Plurality*-#AC$_{\mathrm{D}}$-Control *is* #P*-metric-complete.*

Proof. Let $I = (E, C', c, k)$ be an instance of *Plurality*-#AC$_{\mathrm{C}}$-Control and let s_I be the number of solutions to I, i.e., the number of sets of at most k candidates from C' whose inclusion in election E makes c the unique winner. Note, there are exactly $\sum_{i=0}^{k} \binom{|C'|}{i}$ sets B such that $B \subseteq C'$ and $|B| \leq k$, so we have that the answer to *Plurality*-#AC$_{\mathrm{D}}$-Control for instance I is exactly $\sum_{i=0}^{k} \binom{|C'|}{i} - s_I$.

After theorem 4.1, every #P-problem parsimoniously reduces to *Plurality*-#AC$_{\mathrm{C}}$-Control. We also have that *Plurality*-#AC$_{\mathrm{C}}$-Control metrically reduces to *Plurality*-#AC$_{\mathrm{D}}$-Control, since the number of solutions to I depends on s_I and on I itself. Thus, every #P-problem metrically reduces to *Plurality*-#AC$_{\mathrm{D}}$-Control. □

Let us now move on to control by deleting candidates in plurality elections. The proof of the following theorem is also based on the similar one from [FHH11a].

Theorem 4.3. *Plurality*-#DC$_{\mathrm{C}}$-Control *is* #P*-parsimonious-complete.*

Proof. It is clear that this problem belongs to #P. We prove that it is #P-parsimonious-hard by showing a parsimonious reduction from #X3C.

Let $(B, \mathcal{S})$ be an instance of the #X3C problem, where $B = \{b_1, \ldots, b_{3k}\}$ and $\mathcal{S} = \{S_1, \ldots, S_r\}$ with $k \geq 7$ and $r \geq 1$.

We construct a *Plurality*-#DC$_{\mathrm{C}}$-Control instance as follows. Let $A = \{a_1, \ldots, a_r\}$ and let $E = (C, V)$ be an election, where $C = B \cup A \cup \{c\}$, and collection V contains groups of voters with the following preferences. For each vote we specify up to two top candidates and up to one ranked-lowest candidate; the remaining candidates are ranked in arbitrary order in the '$\cdots$' part.

1. For each set $S_j \in \mathcal{S}$, for each $b_i \in S_j$, we have 1 vote $a_j \succ b_i \succ \cdots \succ c$.
2. For each set $S_j \in \mathcal{S}$, we have 1 vote $a_j \succ c \succ \cdots$.
3. For each $b_i \in B$, we have $k-2$ votes $b_i \succ \cdots \succ c$.

In election E we have the following scores:

1. $score_E(c) = 0$,
2. for each $b_i \in B$, $score_E(b_i) = k-2$, and
3. for each $a_j \in A$, $score_E(a_j) = 4$.

The winners of election E are exactly the candidates in B. We claim that every set A', $A' \subseteq A$, such that $|A'| \leq k$ and that c is a unique winner of plurality election $E' = (C - A', V)$ corresponds one-to-one to a family $\mathcal{S}'$, $\mathcal{S}' \subseteq \mathcal{S}$, such that $|\mathcal{S}'| = k$ and $\mathcal{S}'$ is an exact set cover of B.

First, assume that A' is a subset of A such that $\{S_i \in \mathcal{S} : a_i \in A'\}$ is an exact set cover of B. It is easy to see that c is a unique winner of plurality election $E' = (C - A', V)$, because $score_{E'}(c) = k$, for each $b_i \in B$, $score_{E'}(b_i) = k - 1$, and for each $a_j \in A - A'$, $score_{E'}(a_j) = 4$.

On the other hand, assume that there exists a set $A' \subseteq B \cup A$ of candidates, $|A'| \leq k$, such that c is a unique winner of election $E' = (C - A', V)$. Since $|A'| \leq k$, there are at least $2k$ candidates from B in E' and so it has to be $score_{E'}(c) \geq k - 1$. However, the only way to increase c's score to $k - 1$ (or higher) by deleting at most k candidates is to delete $k - 1$ (or more) candidates from A. Now, if we delete exactly $k - 1$ candidates from A, there is some candidate $b_i \in B$ in the election, for which $score_{E'}(b_i) \geq k - 1$. Thus, A' must contain exactly k candidates from A. Deleting these candidates increases c's score to be k. To ensure that the scores of the candidates in B do not exceed k, we must ensure that A' corresponds to an exact set cover of B by sets from $\mathcal{S}$, so the score of each candidate in B chops down to at most $k - 1$. □

Theorem 4.4. *Plurality-#DC*$_{\text{D}}$*-*Control *is #P-metric-complete.*

Proof. This fact follows from #P-parsimonious-completeness of constructive version of deleting candidates in plurality election. Similarly as in theorem 4.2, we define $I = (E, c, k)$ to be an instance of *Plurality*-#DC$_{\text{C}}$-Control and we define s_I to be a number of solutions to I. It can be verified that the answer to *Plurality*-#DC$_{\text{D}}$-Control for I is exactly $\sum_{i=0}^{k} \binom{m-1}{i} - s_I$. Thus, the problem is #P-metric-complete, after the similar argumentation as in theorem 4.2. □

4.2. Adding and Deleting Voters

In control by adding voters we have a collection V' containing additional voters we can use to make c the unique winner. Intuitively, in order to make c the unique winner, we should add to the election voters from V' that rank c first, and once we have done that, we can get from V' other voters, while not exceeding k selections in total. This strategy leads us to a polynomial-time algorithm which returns the number of such selections.

Theorem 4.5. *Plurality-#AV*$_{\text{C}}$*-*Control *is in* FP.

Proof. For each integer i, $0 \leq i \leq m - 1$, let A_i be a multiset, such that $A_i \subseteq V'$ and it contains voters that rank c_i first (we assume $c_0 = c$). We provide the algorithm which counts all the solutions, and we prove it works in polynomial time. We use function $count(E, V', c, k, j)$ which counts how many possibilities there are, such that we can choose at most $k - j$ elements from $V' - A_0$, preserving c as the unique winner of the altered election. The pseudocode of the algorithm is presented below.

```
Pl-AVC-Count($E, V', c, k$)
1  if $c$ is the unique winner of $E$
2      then $k_0 := 0$
3      else  $k_0 := \max_{c_i \in C}(score_E(c_i) - score_E(c) + 1)$
4  $result := 0$
5  for $j := k_0$ to $\min(|A_0|, k)$
6          do $result := result + \binom{|A_0|}{j} \cdot count(E, V', c, k, j)$
7  return $result$
```

At the beginning, the algorithm finds the minimum number k_0 of elements from A_0 that added to V will make c the unique winner of election E. Clearly, if c already is the unique winner of E, then k_0 is 0, and otherwise k_0 is $\max_{c_i \in C}(score_E(c_i) - score_E(c) + 1)$. After we compute k_0, we loop from this value until $|A_0|$ or k is reached, depending which of them is smaller, and we collect partial results. More specifically, for each integer j, $k_0 \le j \le \min(|A_0|, k)$, we compute the number of multisets U, $U \subseteq V'$, such that U contains exactly j voters from A_0, at most $k - j$ voters from $V' - A_0$, and c is the unique winner of $E = (C, V \uplus U)$. It is easy to verify that for a given j, there are exactly $h(j) = \binom{|A_0|}{j} \cdot count(E, V', c, k, j)$ such multisets. Our algorithm returns $\sum_{j=k_0}^{\min(|A_0|,k)} h(j)$. To complete the proof it suffices to show the subroutine *count* can be computed in polynomial time.

Let us fix j, $k_0 \le j \le \min(|A_0|, k)$ and show how to compute $count(E, V', c, k, j)$. Our goal is to count the number of ways in which we can add at most $k - j$ voters from $V' - A_0$ so that no candidate $c_i \in C$ has score higher than $score_E(c) + j - 1$. For each candidate $c_i \in C$, we can add at most

$$\ell_i = \min\bigl(|A_i|, j + score_E(c) - score_E(c_i) - 1\bigr),$$

voters from A_i; otherwise c_i's score would exceed $score_E(c) + j - 1$.

For each integer p, $0 \le p \le k - j$, and each integer q, $1 \le q \le m - 1$, let $a_{p,q}$ be the number of multisets U, $U \subseteq A_1 \cup A_2 \cup \cdots \cup A_q$, that contain exactly p voters and such that each candidate $c_1, c_2, \ldots, c_q$ has score at most $score_E(c) + j - 1$ in the election $E = (C, V \uplus U)$. It can be checked that $a_{p,q}$ holds the recursion:

$$a_{p,q} = \begin{cases} \sum_{i=0}^{\min(\ell_q,p)} \binom{|A_q|}{i} a_{p-i,q-1}, & \text{if } p > 0 \text{ and } q > 1, \\ 1, & \text{if } p = 0 \text{ and } q > 1, \\ \binom{|A_1|}{p}, & \text{if } p \le |A_1| \text{ and } q = 1, \\ 0, & \text{if } p > |A_1| \text{ and } q = 1. \end{cases}$$

These values can be computed in polynomial time via standard dynamic programming method. Naturally, $count(E, V', c, k, j) = \sum_{p=0}^{k-j} a_{p,m-1}$.

As we can see, function *count* is polynomial-time computable, and so is the main algorithm. □

Using this theorem, we can easily get a result for the destructive setting.

Theorem 4.6. *Plurality*-#$\mathrm{AV_D}$-Control *is in* FP.

Proof. The main observation we make, is that the solution to *Plurality*-#$\mathrm{AV_D}$-Control is equal to the number of all possible selections of at most k new voters from V', subtracted by the number of such selections that make c the unique winner. There are exactly $\sum_{i=0}^{k}\binom{|V'|}{i}$ multisets U such that $U \subseteq V'$ and $|U| \leq k$. Of these, there are exactly Pl-AVC-Count(E, V', c, k) multisets of voters whose inclusion in the election ensures that c is the unique winner. Thus, there are exactly

$$\sum_{i=0}^{k}\binom{|V'|}{i} - \text{Pl-AVC-Count}(E, V', c, k)$$

multisets U of cardinality at most k containing voters from V' whose inclusion in the election ensures that c is not the unique winner. Clearly, we can compute this value in polynomial time. □

The control by deleting voters is similar to the one where we add them, but instead of looking for additional voters, we can only remove them from the election. It is clear that deleting voters for which c is not the top-ranked candidate is a necessary action to make c the winner. We use this observation in the following theorem.

Theorem 4.7. *Plurality*-#$\mathrm{DV_C}$-Control *is in* FP.

Proof. For each integer i, $0 \leq i \leq m-1$, let A_i be a multiset, such that $A_i \subseteq V$ and it contains voters that rank c_i first (we assume $c_0 = c$). For each integer j, $0 \leq j \leq k$, we define $count(E, c, k, j)$ to be the number of multisets U, $U \subseteq V - A_0$ such that:

1. $|U| \leq k - j$, and
2. in election $E' = (C, V - U)$ for each candidate $c_i \in C$ it holds $score_{E'}(c_i) \leq score_E(c) - j - 1$.

The following algorithm returns the number of solutions to the problem.

```
Pl-DVC-Count($E, c, k$)
1  $result := 0$
2  for $j := 0$ to $\min(|A_0|, k)$
3          do $result := result + \binom{|A_0|}{j} \cdot count(E, c, k, j)$
4  return $result$
```

In each iteration of the main loop we consider deleting exactly j voters from A_0 (there are $\binom{|A_0|}{j}$ ways to pick these j voters). Assuming we remove from V exactly j members of A_0, we must also remove some number of voters from $V - A_0$, to make sure that c is the unique winner of the resulting election. The number of ways in which this can be achieved is $count(E, c, k, j)$. It is easy to verify that indeed our algorithm works correctly. It remains to show how to compute $count(E, c, k, j)$.

Let us fix some value j, $0 \leq j \leq \min(|A_0|, k)$. For each $c_i \in C$, we define:

$$\ell_i = \max(0, j + \mathit{score}_E(c_i) - \mathit{score}_E(c) + 1).$$

Intuitively, ℓ_i is the minimal number of voters from A_i that need to be removed from the election for c to have score higher than c_i (assuming j voters from A_0 have been already removed from the election).

For each integer p, $0 \leq p \leq k - j$, and each integer q, $1 \leq q \leq m - 1$, let $a_{p,q}$ be the number of multisets U, $U \subseteq A_1 \cup A_2 \cup \dots \cup A_q$, such that $|U| = p$ and each candidate c_1, $c_2, \dots, c_q$ has score at most $\mathit{score}_E(c) - j - 1$ in election $E' = (C, V - U)$. The following recursive relation holds:

$$a_{p,q} = \begin{cases} \sum_{i=\ell_q}^{\min(|A_q|,p)} \binom{|A_q|}{i} a_{p-i,q-1}, & \text{if } p \geq \ell_q \text{ and } q > 1, \\ \binom{|A_1|}{p}, & \text{if } p \geq \ell_1 \text{ and } q = 1, \\ 0, & \text{if } p < \ell_q. \end{cases}$$

Thus, for each p, q we can compute $a_{p,q}$ in polynomial time using dynamic programming techniques in polynomial time. It is easy to see that $\mathit{count}(E, c, k, j) = \sum_{p=0}^{k-j} a_{p,m-1}$, which also is polynomial-time computable. □

As earlier, we can show that the destructive control by deleting voters can be considered as a complement to the number of possibilities in the constructive case of this problem.

Theorem 4.8. *Plurality*-#DV$_{\text{D}}$-CONTROL *is in* FP.

Proof. We will show it by analogy to the proof of theorem 4.6. The destructive case of deleting voters can be solved by computing the number of possible selections of at most k voters, which will be removed from the election in order to make c the unique winner, and by subtracting this value from the number of all possible selections of at most k voters. There are exactly $\sum_{i=0}^{k} \binom{n}{i}$ multisets U such that $U \subseteq V$ and $|U| \leq k$. Among them there are exactly PL-DVC-COUNT(E, c, k) multisets of voters whose deletion from the election ensures that c is the unique winner. Therefore, the answer to this problem is

$$\sum_{i=0}^{k} \binom{n}{i} - \text{PL-DVC-COUNT}(E, c, k),$$

and it can obviously be produced in polynomial time. □

4.3. Summary

In this chapter we have studied counting variants of control problems under plurality voting system. We showed that controlling by altering the set of candidates leads to #P-complete problems—control by adding or deleting candidates when considering the constructive case is #P-parsimonious-complete, while under the destructive case these types of control are #P-metric-complete. However, each type of control involving changes in the voters' multiset is computationally easy—we showed that these problems are in FP and we developed algorithms that solve them in polynomial-time.

Chapter 5

Control Problems in Approval Voting and in Condorcet Voting

In the current chapter we study both approval voting and Condorcet voting. While these two systems are different in many aspects, their behavior with respect to election control is very similar. Specifically, for both systems counting variants of control problems focusing on modifications of the collection of voters are #P-complete. For both systems it is impossible to make some candidate a winner by adding candidates, and for both systems it is impossible to prevent someone from winning by deleting candidates. Because of these similarities we analyze algorithms for control problems in approval system and then adapt them for Condorcet system.

Approval voting can be considered as a generalization of plurality voting because it allows voters to specify not only the single candidate which they prefer most, but any number of candidates they like. This change makes this method vulnerable to some types of control, while to the others it becomes resistant or even immune. More specifically, approval voting is vulnerable to constructive control by deleting candidates, to destructive control by adding candidates, and to destructive adding or deleting voters. It turns out that, unlike in plurality voting, the counting variants of control problems where we modify the set of candidates are in FP, and the counting variants of control problems where we modify the collection of voters are #P-complete. We show these facts formally in the following sections.

5.1. Adding and Deleting Candidates

Both approval and Condorcet have been proven to be immune to control by adding candidates in constructive case. Formally speaking, the answer to their counting variants is always 0, thus the counting problems belong to FP. So do their destructive cases (although it is not completely trivial), as we show in the following theorem.

Theorem 5.1. *Approval-*#AC$_\text{D}$-Control *and Condorcet-*#AC$_\text{D}$-Control *are in* FP.

Proof. First, let us consider approval system. Since we assume every voter $v_i \in V$ provides a set A_i of approved candidates, $A_i \subseteq C \cup C'$, we can easily determine which $c' \in C'$ would beat c, when added to the election E. Let A be the set of candidates in

C' that are approved of by at least as many voters as c is. The number of possibilities depends on the fact, whether c is already the unique winner of E. If c is the unique winner prior to adding any candidates, then it suffices to include at least one candidate from A to the election to ensure that c is not the unique winner. Otherwise, we can pick any set of k or less candidates from C' and include them in the election.

This observation leads us to the following algorithm.

```
AP-ACD-COUNT($E, C', c, k$)
1  if $c$ is not the unique winner of $E$
2      then return $\sum_{i=0}^{k} \binom{|C'|}{i}$
3  Let $A$ be the set of candidates $c' \in C'$, s.t. $score_{(C \cup C', V)}(c') \geq score_{(C \cup C', V)}(c)$.
4  $result := 0$
5  for $j := 1$ to $k$
6      do $result := result + \sum_{i=1}^{\min(|A|,j)} \binom{|A|}{i}\binom{|C'-A|}{j-i}$
7  return $result$
```

The loop from line 5, for every j, counts the ways in which we can choose exactly j candidates from C'; it can be done by first picking i of the candidates in A (who beat c), and then $j - i$ of the candidates in $C' - A$. (If $j - i > |C' - A|$, then $\binom{|C'-A|}{j-i} = 0$ by the definition of binomial coefficient.)

It is clear that the algorithm works in polynomial time, so the problem *Approval*-#AC$_{\text{D}}$-CONTROL is in FP.

The procedure we presented for *Approval*-#AC$_{\text{D}}$-CONTROL can be easily transformed to the one for *Condorcet*-#AC$_{\text{D}}$-CONTROL. Note that by adding to the election only one candidate $c' \in C'$ such that $N_{E'}(c, c') \leq n/2$, we can preclude c from being the Condorcet winner. Thus, the only changes compared to algorithm AP-ACD-COUNT are:

1. In the first line, instead of testing if c is the unique winner of approval election E we need to test if c is a Condorcet winner of E, and
2. In the line 3 we redefine the set A to be the set of candidates $c' \in C'$ such that $N_{E'}(c, c') \leq n/2$.

Of course, the algorithm we received after this transformation runs in polynomial time, so *Condorcet*-#AC$_{\text{D}}$-CONTROL $\in$ FP. $\square$

Now let us consider a problem dual to the previous one—we now want to delete candidates to either make the distinguished one the unique winner, or to preclude him from being the unique winner. We only show how to solve constructive case of this type of control, because approval and Condorcet are immune in its destructive case (which means that both *Approval*-#DC$_{\text{D}}$-CONTROL and *Condorcet*-#DC$_{\text{D}}$-CONTROL are trivially in FP).

Theorem 5.2. *Approval*-#DC$_{\text{C}}$-CONTROL *and Condorcet*-#DC$_{\text{C}}$-CONTROL *are in* FP.

Proof. The only way to make $c \in C$ the unique winner of approval election E, is to remove from $C - \{c\}$ such candidates c', for which $score_E(c') > score_E(c)$. They can be found immediately, since we know how to efficiently compute scores of the candidates. Let us assume there are k_0 such candidates. After removing all of them from $C - \{c\}$, we can also remove $k - k_0$ or less of any remaining candidates other than c. Based on this observation we provide the following simple algorithm.

```
AP-DCC-COUNT($E, c, k$)
1  Let $k_0$ be the number of candidates $c' \in C - \{c\}$, s.t. $score_E(c') \geq score_E(c)$.
2  return $\sum_{i=0}^{k-k_0} \binom{m-k_0-1}{i}$
```

Clearly, the algorithm runs in polynomial-time.

We receive the procedure for *Condorcet*-#DC$_C$-Control after redefining k_0 to be the number of candidates c' in $C - \{c\}$ for which $N_E(c, c') \leq n/2$. The reader can easily verify that this modification leads to the correct polynomial-time algorithm. □

5.2. Adding and Deleting Voters

Let us now move on to the control problems that modify the collection of voters. Each of four types of counting problems considered in this section is #P-complete, so instead of giving algorithms, we prove these facts formally by showing the particular types of #P-completeness for each of them.

Theorem 5.3. *Approval*-#AV$_C$-Control *is #P-parsimonious-complete.*

Proof. It is clear that this problem is in #P. In order to prove its #P-parsimonious-hardness we show a parsimonious reduction from #X3C.

Given an instance $(B, \mathcal{S})$ of #X3C problem, where $B = \{b_1, \ldots, b_{3k}\}$ and $\mathcal{S} = \{S_1, \ldots, S_r\}$, we construct the following instance of *Approval*-#AV$_C$-Control. Let $E = (C, V)$ be an election in which $C = B \cup \{c\}$, where c is a distinguished candidate, and V consists of $k - 2$ registered voters who each approve of $b_1, \ldots, b_{3k}$ and disapprove of c. We have also a multiset V' that consists of r unregistered voters. For each set $S_j \in \mathcal{S}$, there is a voter in V' who approves of c and the 3 candidates in S_j, and who disapproves of all other candidates.

We claim that every multiset U, $U \subseteq V'$, such that $|U| \leq k$ and that c is a unique winner of approval election $E' = (C, V \uplus U)$ corresponds one-to-one to a family $\mathcal{S}'$, $\mathcal{S}' \subseteq \mathcal{S}$, such that $|\mathcal{S}'| = k$ and $\mathcal{S}'$ is an exact set cover of B.

First, let us assume that there is an exact set cover of B in $\mathcal{S}$. We fix U to be the whole V' and we consider an election $E' = (C, V \uplus V')$ which, compared to E, has the k additional voters that correspond to an exact set cover of B. Then we have $score_{E'}(c) = k$, and for every $b_i \in B$, we have $score_{E'}(b_i) = k - 1$, so c is the unique winner of E'.

Now suppose there is a multiset U, $U \subseteq V'$, containing at most k voters, such that c is the unique winner of election $E' = (C, V \uplus U)$. Then we clearly need to have $|U| = k - 1$, but then $|U| = k$, as some $b_i \in B$ reaches additional points. Every $b_i \in B$ can gain at most 1 point. Since each voter in V' approves of exactly 3 candidates from B, it follows that every $b_i \in B$ gains exactly 1 point. Thus, multiset U corresponds to an exact set cover of B. $\square$

Theorem 5.4. *Condorcet-*$\#\mathrm{AV_C}$-Control *is* $\#$P*-parsimonious-complete.*

Proof. The problem is clearly in #P, so it suffices to show that it is #P-parsimonious-hard. We show the transformation from #X3C.

Let $(B, \mathcal{S})$ be an instance of #X3C problem, where $B = \{b_1, \ldots, b_{3k}\}$ and $\mathcal{S} = \{S_1, \ldots, S_r\}$. We create an election $E = (C, V)$, where $C = B \cup \{c\}$. Let V consist of $k-3$ voters with preferences $b_1 \succ b_2 \succ \cdots \succ b_{3k} \succ c$. Thus b_1 is the Condorcet winner of E, and every candidate $b_i \in B$ beats c in $k-3$ votes.

For each set $S_j \in \mathcal{S}$, let V' contain a voter with preference order $b_{j_1} \succ b_{j_2} \succ b_{j_3} \succ c \succ \cdots$, where $b_{j_1}, b_{j_2}, b_{j_3} \in S_j$ (after c the remaining candidates are ranked in arbitrary order). We claim that every multiset U, $U \subseteq V'$, such that $|U| \leq k$ and that c is a Condorcet winner of election $E' = (C, V \uplus U)$ corresponds one-to-one to a family $\mathcal{S}'$, $\mathcal{S}' \subseteq \mathcal{S}$, such that $|\mathcal{S}'| = k$ and $\mathcal{S}'$ is an exact set cover of B.

First assume that $\mathcal{S}' \subseteq \mathcal{S}$ is an exact set cover of B. For each $S_j \in \mathcal{S}'$ we include the corresponding voter from V' to multiset U and we consider an election $E' = (C, V \uplus U)$. For each $b_i \in B$ we have $N_{E'}(b_i, c) = N_E(b_i, c) + 1 = k - 2$ and $N_{E'}(c, b_i) = N_E(c, b_i) + k - 1 = k - 1$. Thus c becomes the Condorcet winner of E'.

Now assume that c is the Condorcet winner in election $E' = (C, V \uplus U)$, where $U \subseteq V'$. There cannot be more than 1 voter in U who prefers any $b_i \in B$ to c, since then we would have $N_{E'}(b_i, c) \geq N_E(b_i, c) + 2 = k-1$, and $N_{E'}(c, b_i) \leq N_E(c, b_i) + k - 2 = k-2$, and so c would lose to b_i. Thus each b_i is preferred to c by either 0 or 1 voters from U. If b_i is preferred by 1 voter from U, then for c to win he must be preferred by $k-1$ voters from U, and since some voter must be added, there must be $|U| = k$.

If there are no voters in U who prefer b_i to c, then since the preferences of the k new voters each include 3 positions above c, there must be some other $b_{i'}$ that is ranked above c by more than 1 voter. This contradicts the requirement that no more than 1 voter in U prefers any other candidate to c. Therefore each b_i is preferred to c by exactly 1 of the k voters in U. Thus, the voters from U correspond to an exact set cover of B. $\square$

Theorem 5.5. *Approval-*$\#\mathrm{AV_D}$-Control *and Condorcet-*$\#\mathrm{AV_D}$-Control *are* $\#$P*-metric-complete.*

Proof. The answer to problem *Approval-*$\#\mathrm{AV_D}$-Control is the total number of ways we can choose at most k new voters from V' (which are exactly $\sum_{i=0}^{k} \binom{|V'|}{i}$)) subtracted by the value returned by the problem *Approval-*$\#\mathrm{AV_C}$-Control. After theorem 5.3, *Approval-*$\#\mathrm{AV_C}$-Control is #P-parsimonious-complete, and *Approval-*$\#\mathrm{AV_C}$-Control metrically reduces to *Approval-*$\#\mathrm{AV_D}$-Control, so *Approval-*$\#\mathrm{AV_D}$-Control is #P-metric-complete.

The similar argumentation holds for problem *Condorcet-*$\#\mathrm{AV_D}$-Control. The solution is equal to $\sum_{i=0}^{k} \binom{|V'|}{i}$ subtracted by the value of *Condorcet-*$\#\mathrm{AV_C}$-Control for the given instance. Therefore, this problem is #P-metric-complete as well. $\square$

Theorem 5.6. *Approval-#*$\mathrm{DV_C}$-CONTROL *is* #P*-parsimonious-complete.*

Proof. This problem is in class #P. We show it is also #P-parsimonious-hard by parsimonious reduction from the problem #X3C.

Suppose we are given an instance of #X3C, $(B, \mathcal{S})$, where $B = \{b_1, \dots, b_{3k}\}$, $\mathcal{S} = \{S_1, \dots, S_r\}$. For each $b_i \in B$, we set ℓ_i to be the number of sets in $\mathcal{S}$ containing b_i.

Let $E = (C, V)$ be an election in which $C = B \cup \{c\}$, where c is the distinguished candidate, and collection V consists of the following voters:

1. We have voters u_1, u_2, …, u_r such that, for each integer j, $1 \le j \le r$, u_j approves of all candidates in S_j and u_j disapproves of all other candidates.
2. We have voters v_1, v_2, …, v_r such that, for each integer j, $1 \le j \le r$, v_j approves of c, and v_j approves of b_i if and only if $j \le r - \ell_i$.

Note that in the election E each candidate has exactly r points.

We claim that every multiset U, $U \subseteq V$, such that $|U| \le k$ and that c is a unique winner of approval election $E' = (C, V - U)$ corresponds one-to-one to a family $\mathcal{S}'$, $\mathcal{S}' \subseteq \mathcal{S}$, such that $|\mathcal{S}'| = k$ and $\mathcal{S}'$ is an exact set cover of B.

Suppose there is an exact set cover of B in $\mathcal{S}$ and let U, $U \subseteq \{u_1, \dots, u_r\} \subseteq V$, be a multiset of k voters that correspond to this set cover. In an election $E' = (C, V - U)$ every $b_i \in B$ loses one point, leaving c the unique winner of E'.

Now suppose there is a multiset U, $U \subseteq V$, containing at most k voters, such that c is the unique winner of election $E' = (C, V - U)$. Suppose that for some j, $1 \le j \le r$, U contains v_j. Then c has less than r points and there are $b_i \in B$ who still have r points each, and others who have $r - 1$ points each. In order to make c the unique winner, we need to remove additional voters, so that candidates from B lose more than $3k$ points in total. Because each voter u_j approves of exactly 3 candidates from B, it contradicts the fact that $|U| \le k$. Thus, we assume that $U \subseteq \{u_1, \dots, u_r\}$. For c to have become the unique winner, every $b_i \in B$ must have lost at least 1 point. It follows that the deleted voters correspond to a set cover of B, and since the cover has size at most k, this must be an exact set cover of B. ☐

Theorem 5.7. *Condorcet-#*$\mathrm{DV_C}$-CONTROL *is* #P*-parsimonious-complete.*

Proof. It is clear that this problem belongs to #P. We show that the problem is #P-parsimonious-hard by transformation from #X3C.

Suppose we are given an instance of #X3C, $(B, \mathcal{S})$, where $B = \{b_1, \dots, b_{3k}\}$, $\mathcal{S} = \{S_1, \dots, S_r\}$. Without loss of generality, we assume that $r \ge k$ and $k > 2$ (if $r < k$ then $\mathcal{S}$ does not contain a set cover of B, and if $k \le 2$ then we can solve the problem by brute force).

We create an election $E = (C, V)$ in which $C = B \cup \{c, d\}$ and V is the following collection of $4r - k + 1$ voters ('$\cdots$' part denotes the remaining candidates in arbitrary order):

1. We have $r - k + 2$ voters with preference $c \succ d \succ \cdots$.
2. We have $r - 1$ voters with preference $\cdots \succ c \succ d$.

3. For each $S_j \in \mathcal{S}$ ($S_j = \{b_{j1}, b_{j2}, b_{j3}\}$) we have two voters u_j and v_j, such that
   a) u_j has preference $d \succ \cdots \succ c \succ b_{j1} \succ b_{j2} \succ b_{j3}$,
   b) v_j has preference $d \succ b_{j1} \succ b_{j2} \succ b_{j3} \succ c \succ \cdots$.

It is easy to see that for all $b_i \in B$, $N_E(c, b_i) = 2r - k + 2$, and $N_E(c, d) = 2r - k + 1$.

We claim that every multiset U, $U \subseteq V$, such that $|U| \leq k$ and that c is a Condorcet winner of election $E' = (C, V - U)$ corresponds one-to-one to a family $\mathcal{S}'$, $\mathcal{S}' \subseteq \mathcal{S}$, such that $|\mathcal{S}'| = k$ and $\mathcal{S}'$ is an exact set cover of B.

If family $\mathcal{S}$ contains a k-element set cover of B, say $\{S_{a_1}, \ldots, S_{a_k}\}$, then set $U = \{u_{a_1}, \ldots, u_{a_k}\}$. In election $E' = (C, V - U)$ we have $4r - 2k + 1$ voters, for each $b_i \in B$, $N_{E'}(c, b_i) = N_E(c, b_i) - 1$, and $N_{E'}(c, d) = N_E(c, d)$. Thus, c is a Condorcet winner of E'.

Now assume there is a multiset U, $U \subseteq V$, of k or fewer voters, such that c is the Condorcet winner in election $E' = (C, V - U)$. The value of $N_{E'}(c, d)$ is the same as $N_E(c, d)$, because we cannot increase it when deleting voters. Thus, c defeats d only if $|U| = k$. Furthermore, for each $b_i \in B$, $N_{E'}(c, b_i) \geq 2r - k + 1$, so in U we have at most one voter who prefers c to b_i, and since $|U| = k$, we have exactly one such voter. It follows, that $U \subseteq \{u_1, \ldots, u_r\}$, and that voters in U correspond to an exact set cover of B. □

Theorem 5.8. *Approval-*#DV$_{\text{D}}$-Control *and Condorcet-*#DV$_{\text{D}}$-Control *are* #P*-metric-complete.*

Proof. We base on the idea from theorem 5.5—the problem *Approval*-#DV$_{\text{D}}$-Control is a "complement" to *Approval*-#DV$_{\text{C}}$-Control, and the problem *Condorcet*-#DV$_{\text{D}}$-Control is a "complement" to *Condorcet*-#DV$_{\text{C}}$-Control. The only change compared to theorem 5.5 is that, instead of $|V'|$, we have $|V|$ in the sum for total number of choices. Thus, both problems are #P-metric-complete. □

5.3. Summary

The main conclusion of this chapter is that approval and Condorcet systems share many similarities with respect to election control. Counting variants of each type of control in approval is as hard as in Condorcet. Moreover, the algorithms that solve a certain type of control in one system require minor changes only to make them appropriate for the other system.

Another observation one can make is that as far as the complexity of election control goes, approval and Condorcet behave in an exactly opposite way to plurality. In plurality, every control problem involving modification of candidate set is #P-complete, while such types of control under approval and Condorcet systems are in FP. Similarly, control by adding or deleting voters in plurality is #P-complete, while in approval and Condorcet it is in FP.

Chapter 6

Control Problems in Maximin Voting

Maximin is the last voting system we study in this thesis. The proofs in this chapter are motivated by the analysis of maximin in the work of Faliszewski, Hemaspaandra, and Hemaspaandra [FHH11b].

Several proofs in this chapter use an extended notation for total orders. Let $X = \{x_1, x_2, \ldots, x_k\}$ be a set and $\mathcal{T}(X)$ be a set of all total orders over X. For a fixed total order $T_X \in \mathcal{T}(X)$ such that $x_1 \succ x_2 \succ \cdots \succ x_k$ we write $y \succ T_X \succ z$ to denote $y \succ x_1 \succ x_2 \succ \cdots \succ x_k \succ z$. Also, we define $\overleftarrow{T_X}$ to be a total order T_X but in reverse, i.e., $x_k \succ x_{k-1} \succ \cdots \succ x_1$.

6.1. Adding and Deleting Candidates

In the following theorem we show #P-completeness of the counting variant of constructive adding candidates and we use the result in the next theorem concerning the destructive case.

Theorem 6.1. *Maximin*-#AC$_{\mathrm{C}}$-Control *is* #P*-parsimonious-complete.*

Proof. It is clear that this problem belongs to #P. We give a parsimonious reduction from #X3C. Let $(B, \mathcal{S})$, where $B = \{b_1, \ldots, b_{3k}\}$ and $\mathcal{S} = \{S_1, \ldots, S_r\}$, be an instance of the #X3C problem.

We construct a *Maximin*-#AC$_{\mathrm{C}}$-Control instance as follows. Let $E = (C, V)$ be an election, where $C = B \cup \{c\}$ is the set of registered candidates, and $V = \{v_1, \ldots, v_{2r+2}\}$ is a multiset of voters. We also have a set $A = \{a_1, \ldots, a_r\}$ of spoiler (unregistered) candidates. Each candidate a_i in A corresponds to a set S_i in $\mathcal{S}$. For each $S_i \in \mathcal{S}$ we fix total orders $T_{S_i} \in \mathcal{T}(S_i)$, $T_{B-S_i} \in \mathcal{T}(B - S_i)$, and $T_{A-\{a_i\}} \in \mathcal{T}(A - \{a_i\})$. For each $S_i \in \mathcal{S}$, voter v_i reports preference order $c \succ T_{B-S_i} \succ a_i \succ T_{S_i} \succ T_{A-\{a_i\}}$ and voter v_{r+i} reports preference order $\overleftarrow{T_{A-\{a_i\}}} \succ a_i \succ \overleftarrow{T_{S_i}} \succ \overleftarrow{T_{B-S_i}} \succ c$. Voter v_{2r+1} reports $c \succ T_A \succ T_B$ and voter v_{2r+2} reports $\overleftarrow{T_B} \succ c \succ \overleftarrow{T_A}$, where $T_A \in \mathcal{T}(A)$, and $T_B \in \mathcal{T}(B)$ are some fixed total orders.

We claim that every set A', $A' \subseteq A$, such that $|A'| \leq k$ and that c is a unique winner of maximin election $E' = (C \cup A', V)$ corresponds one-to-one to a family $\mathcal{S}'$, $\mathcal{S}' \subseteq \mathcal{S}$, such that $|\mathcal{S}'| = k$ and $\mathcal{S}'$ is an exact set cover of B.

To show the claim, let us observe that for each pair of distinct elements b_i, $b_j \in B$, we have $N_E(b_i, b_j) = N_E(c, b_i) = N_E(b_i, c) = r+1$. That is, the winners in election E are all candidates as all of them tie. Candidate c is a winner of E, but we need to make him the unique winner. Now consider a set $A' \subseteq A$, $|A'| \leq k$, and an election $E' = (C \cup A', V)$. Note that values of N_E and $N_{E'}$ are the same for each pair of candidates in $B \cup \{c\}$. For each pair of distinct elements a_i, $a_j \in A'$, we have $N_{E'}(c, a_i) = r+2$, $N_{E'}(a_i, c) = r$, and $N_{E'}(a_i, a_j) = r+1$. For each $b_i \in B$ and each $a_j \in A'$ we have that

$$N_{E'}(b_i, a_j) = \begin{cases} r, & \text{if } b_i \in S_j, \\ r+1, & \text{if } b_i \notin S_j, \end{cases}$$

and $N_{E'}(a_j, b_i) = |V| - N_{E'}(b_i, a_j) = 2r+2-N_{E'}(b_i, a_j)$. Thus, by definition of maximin, we have the following scores in E':

1. $\mathit{score}_{E'}(c) = r+1$,
2. for each $a_j \in A'$, $\mathit{score}_{E'}(a_j) = r$, and
3. for each $b_i \in B$,

$$\mathit{score}_{E'}(b_i) = \begin{cases} r, & \text{if there is } a_j \in A' \text{ such that } b_i \in S_j, \\ r+1, & \text{otherwise.} \end{cases}$$

Since $|A'| \leq k$, it is easy to see that c is the unique winner of E' if and only if for each $b_i \in B$, $\mathit{score}_{E'}(b_i) = r$, which holds if and only if family $\mathcal{S}'$ corresponding to A' is an exact set cover of B. □

Theorem 6.2. *Maximin-*#AC$_\text{D}$-Control *is* #P*-metric-complete.*

Proof. We proof if this theorem is analogous to the proof of Theorem 4.2. The answer to this problem is $\sum_{i=0}^{k} \binom{|C'|}{i}$ subtracted by the answer to problem *Maximin-*#AC$_\text{C}$-Control for the same instance. Since the constructive version is #P-parsimonious-complete, the current problem is #P-metric-complete. □

Control by deleting candidates in maximin has an interesting property. While the decision variant of this problem is polynomial-time computable, it is very likely that its counting variant is computationally hard. We were not able to prove this fact, but we simplify *Maximin-*#DC$_\text{C}$-Control by modelling it as a graph problem with a simpler structure.

First we prove that the decision variant of control by deleting candidates in maximin election is in P.

Theorem 6.3. *Maximin-*DC$_\text{C}$-Control *is in* P.

Proof. Given an election $E = (C, V)$, a distinguished candidate $c \in C$, and a positive integer $k \leq m$, we ask if there is a set $B \subseteq C$ with $|B| \leq k$, such that c is the unique winner of election $E' = (C - B, V)$. Let us study how do scores of candidates change after removing candidates. Since score of candidate $p \in C$ in maximin election is defined as $\min_{p' \in C - \{p\}} N_E(p, p')$, we cannot decrease p's score. So, there are two ways to make c the unique winner:

1. by deleting such candidates c' that have greater score than c has, or
2. by deleting such candidates c'' for which $N_E(c, c'') = score_E(c)$, i.e., the value $N_E(c, c'')$ is minimal for c.

Of course, when trying to increase c's score by applying these rules, we may increase other candidates' scores as well, and make these candidates winners instead of c. The idea is to performing these operations while preserving c's score, and then to check if the number of deleted candidates does not exceed k.

Based on this observation we create a polynomial-time algorithm that solves problem *Maximin*-DC$_\text{C}$-Control.

```
MM-DCC($E, c, k$)
1  for $i := 0$ to $n$
2        do Let $M_{c,i}$ be the set of candidates $c' \in C - \{c\}$, s.t. $N_E(c, c') = i$.
3  for $j := score_E(c)$ to $\max_{c' \in C - \{c\}} N_E(c, c')$
4        do $B := \bigcup_{i=0}^{j-1} M_{c,i}$
5           while $(C - \{c\}) - B$ contains $p$, s.t. $score_{(C-B,V)}(p) \geq score_{(C-B,V)}(c)$
6                do $B := B \cup \{p\}$
7           if $M_{c,j} \not\subseteq B$ and $|B| \leq k$
8             then return "yes"
9  return "no"
```

The first step in the algorithm is computing, for each integer i, $0 \leq i \leq n$, the set $M_{c,i} = \{\, c' \in C - \{c\} : N_E(c, c') = i \,\}$. Note that if $j \leq \max_{c' \in C - \{c\}} N_E(c, c')$ and $B = \bigcup_{i=0}^{j-1} M_{c,i}$, then c have score equal to j in election $(C - B, V)$.

The main loop of the algorithm (lines 3–8) consists of at most $n + 1$ steps. For each integer j, $score_E(c) \leq j \leq \max_{c' \in C - \{c\}} N_E(c, c')$, we check if it is possible to delete at most k candidates and make c the unique winner having score equal to j in the modified election. To do this, we set B to be $\bigcup_{i=0}^{j-1} M_{c,i}$, and then we try to add to B such candidates which prevent c from being the unique winner in election $(C - B, V)$. Clearly, every candidate $p \in (C - \{c\}) - B$ of score higher than or equal to c's score in election $(C - B, V)$ should have been added to B. We repeat this operation for a current set B, because the candidates just added to B may have increased scores of other candidates in $(C - B, V)$. This greedy procedure, implemented in lines 5–6, eventually ends. However, now we could have $M_{c,j} \subseteq B$, which would mean that we have increased c's score, but we assumed it should be constant during this iteration of the main loop. Otherwise, if the cardinality of the set B is at most k, the algorithm returns "yes", as c

is the unique winner of election $E' = (C - B, V)$. After checking every value of j without success the algorithm returns "no", because it is impossible to make c the unique winner.

Clearly, our algorithm can be performed in polynomial time, thus we have that *Maximin*-DC$_{\mathrm{C}}$-CONTROL $\in$ P. $\square$

Let us now move on to the counting variant. Note that when the algorithm described in the proof above is going to return "yes", it has already found one possible way to delete candidates. At this moment we may delete additional candidates from $(C - \{c\}) - B$ as long as c is still the unique winner with score equal to j and the total number of deleted candidates does not exceed k. Our goal is to count such possibilities.

When we delete a candidate from $(C - \{c\}) - B$, other candidate's score may increase and become greater than or equal to j, in which case it should also be deleted to make sure c is still a unique winner. Sometimes, when such a situation occurs, we need to delete not only one candidate but a group of them. More formally, candidate p increases its score to be j or greater after we delete each candidate p' from $\bigcup_{i=0}^{j-1} M_{p,i}$. This situation can be modeled as a directed graph G, in which $C - B$ is the set of vertices, and for each pair of vertices u and v, there is an arc from u to v if and only if $v \in \bigcup_{i=0}^{j-1} M_{u,i}$. Note that in order to make v's score greater than or equal to j (and therefore to make v a new winner), we need to delete every candidate represented by vertex u, such that (u, v) is an arc in graph G.

By *terminal vertex* we will call a vertex u such that there are no arcs leaving u. Note that in our graph G, the only terminal vertex is c. Our problem is to count such subsets of candidates from $(C - \{c\}) - B$ whose cardinality does not exceed $k - |B|$ and whose deletion does not change c's score (which is j). For graph G, it means that we count sets of vertices R, $R \subseteq (C - \{c\}) - B$, such that:

1. $|R| \leq k - |B|$,
2. $M_{c,j} \nsubseteq R$, and
3. c is the only terminal vertex of G'—the induced subgraph of G in which the set of vertices is $(C - B) - R$.

The first condition above is trivial. Without the second condition we could change the score of c, while our goal is to keep this score constant. The last condition is also important—if there would be any other terminal vertex in G', we should have removed it from our graph, as the candidate represented by it has score greater than or equal than c's score.

Note, that we can omit the second condition when counting sets of vertices satisfying these conditions. We create induced subgraph G'' of G in which we removed vertices from $M_{c,j}$ and the smallest set of additional vertices which should be further removed in order to make c the graph's only terminal vertex. The set of additional vertices to remove can be computed by a greedy method from the proof of Theorem 6.3. The answer to our counting problem is the number of sets of vertices of G satisfying the first and the third condition, decreased by the number of sets of vertices of G'' satisfying the first and the third condition.

The following conjectures state that control by deleting candidates in maximin (and, equivalently, the graph problem from the previous paragraphs) is computationally hard, in both constructive and destructive cases.

Conjecture 6.4. *Maximin-*#DC$_\mathrm{C}$-Control *is* #P*-parsimonious-complete.*

Conjecture 6.5. *Maximin-*#DC$_\mathrm{D}$-Control *is* #P*-metric-complete.*

If Conjecture 6.4 is true, the proof of Conjecture 6.5 will be simple adaptation of the proof of Theorem 4.4.

6.2. Adding and Deleting Voters

Theorem 6.6. *Maximin-*#AV$_\mathrm{C}$-Control *is* #P*-parsimonious-complete.*

Proof. Clearly, the problem is in #P. We show a parsimonious reduction from #X3C in order to prove it is #P-parsimonious-hardness.

Given an instance $(B, \mathcal{S})$ of #X3C, where $B = \{b_1, \ldots, b_{3k}\}$ and $\mathcal{S} = \{S_1, \ldots, S_r\}$, we construct an election $E = (C, V)$, where $C = B \cup \{c, w\}$ is the set of candidates, and $V = \{v_1, \ldots, v_{4k}\}$ is a multiset of registered voters. For a fixed total order $T_B \in \mathcal{T}(B)$ there are:

1. $2k$ voters with preference order $w \succ T_B \succ c$,
2. k voters with preference order $c \succ T_B \succ w$, and
3. k voters with preference order $c \succ w \succ T_B$.

We also have a multiset V' of r unregistered voters, where ith voter, $1 \le i \le r$, reports preference order $T_{B-S_i} \succ c \succ T_{S_i} \succ w$, with fixed total orders $T_{S_i} \in \mathcal{T}(S_i)$ and $T_{B-S_i} \in \mathcal{T}(B - S_i)$.

We claim that every multiset U, $U \subseteq V'$, such that $|U| \le k$ and that c is a unique winner of maximin election $E' = (C, V \uplus U)$ corresponds one-to-one to a family $\mathcal{S}'$, $\mathcal{S}' \subseteq \mathcal{S}$, such that $|\mathcal{S}'| = k$ and $\mathcal{S}'$ is an exact set cover of B.

It is easy to verify that for each $b_i \in B$ it holds that $N_E(c, b_i) = N_E(c, w) = 2k$. Similarly, for each $b_i \in B$ it holds that $N_E(w, b_i) = 3k$ and that $N_E(w, c) = 2k$. Thus, $score_E(c) = score_E(w) = 2k$. It is also easy to verify that for each $b_i \in B$ it holds that $score_E(b_i) \le k$.

Let U, $U \subseteq V'$, be a multiset containing at most k voters, and let $E' = (C, V \uplus U)$. For each $b_i \in B$ it holds that $score_{E'}(b_i) \le 2k$. Since each voter in V' ranks w as the least desirable candidate, $score_{E'}(w) = 2k$. Let us now find a score of c. If there exists a candidate $b_i \in B$ such that there is no voter in V' that prefers c to b_i, then $N_{E'}(c, b_i) = 2k$ and thus $score_{E'}(c) = 2k$. Otherwise, $score_{E'}(c) \ge 2k + 1$. Thus, c is a unique winner of E' if and only if U corresponds to an exact set cover of B. □

Theorem 6.7. *Maximin-*#DV$_\mathrm{C}$-Control *is* #P*-parsimonious-complete.*

Proof. It is clear this problem is in #P. We show that it is #P-parsimonious-hard by parsimonious transformation from #X3C.

Let $(B, \mathcal{S})$ be an instance of the #X3C problem, where $B = \{b_1, \dots, b_{3k}\}$ with $k \geq 3$, and $\mathcal{S} = \{S_1, \dots, S_r\}$. We form an election $E = (C, V)$ where $C = B \cup \{c, w\}$ and $V = V' \cup V''$ with $V' = \{v'_1, \dots, v'_{2r}\}$, $V'' = \{v''_1, \dots, v''_{2r-k+2}\}$. For each i, $1 \leq i \leq r$, and for fixed total orders $T_{S_i} \in \mathcal{T}(S_i)$, $T_{B-S_i} \in \mathcal{T}(B - S_i)$, voter v'_i reports preference order $w \succ T_{B-S_i} \succ c \succ T_{S_i}$ and voter v'_{r+i} reports preference order $w \succ \overleftarrow{T_{S_i}} \succ c \succ \overleftarrow{T_{B-S_i}}$. Let $T_B \in \mathcal{T}(B)$ be a fixed total order. Among the voters in V'' there are:

1. 2 voters with preference order $c \succ w \succ T_B$,
2. $r - k$ voters with preference order $c \succ T_B \succ w$, and
3. r voters with preference order $T_B \succ c \succ w$.

We claim that every multiset U, $U \subseteq V$, such that $|U| \leq k$ and that c is a unique winner of maximin election $E' = (C, V - U)$ corresponds one-to-one to a family $\mathcal{S}'$, $\mathcal{S}' \subseteq \mathcal{S}$, such that $|\mathcal{S}'| = k$ and $\mathcal{S}'$ is an exact set cover of B.

It is easy to see that candidates in election E have the following scores:

1. $score_E(w) = 2r$ (because $N_E(w, c) = 2r$ and for each $b_i \in B$, $N_E(w, b_i) = 2r + 2$),
2. $score_E(c) = 2r - k + 2$ (because for each $b_i \in B$, $N_E(c, w) = N_E(c, b_i) = 2r - k + 2$), and
3. for each $b_i \in B$, $score_E(b_i) \leq 2r - k$ (because $N_E(b_i, w) = 2r - k$).

Before we delete any voter, w is the unique winner with $k - 2$ more points than c. We can decrease w's score by at most k points via deleting at most k voters.

Let U, $U \subseteq V$, be a multiset of voters such that c is the unique winner of $E' = (C, V - U)$. We partition U into $U' \cup U''$, where $U' = U \cap V'$ and $U'' = U \cap V''$. We claim that U'' is empty. Let us assume that $U'' \neq \emptyset$ and let $E'' = (C, V - U'')$. Since every voter in V'' prefers c to w, we have that $N_{E''}(c, w) = N_E(c, w) - |U''|$ and thus $score_{E''}(c) \leq score_E(c) - |U''|$. Similarly, assuming $U'' \neq \emptyset$, it is easy to see that $score_{E''}(w) \geq score_E(w) - |U''| + 1$. It holds after the observation that deleting any single member of U'' cannot decrease w's score. That is, we have that $score_{E''}(c) \leq 2r - k + 2 - |U''|$ and $score_{E''}(w) \geq 2r + 1 - |U''|$. So in E'', w has at least $k - 1$ more points than c. Since $|U''| \geq 1$, we can delete at most $k - 1$ voters U' from election E''. But then c will not be a unique winner of E', which is a contradiction.

Thus, U contains members of V' only. Since w is ranked first in every vote in V', deleting voters from U decreases w's score by exactly $|U|$. Further, deleting voters U certainly decreases c's score by at least one point. Thus, after deleting voters U we have:

1. $score_{E'}(w) = 2r - |U|$, and
2. $score_{E'}(c) \leq 2r - k + 2 - 1 = 2r - k + 1$.

In consequence, the only possibility that c is a unique winner after deleting voters U is that $|U| = k$ and we have equality in item 2 above. It is easy to verify that this equality holds if and only if U contains k voters among $v'_1, \dots, v'_r$ that correspond to an exact set cover of B via sets from $\mathcal{S}$ (under the assumption that $k \geq 3$). ☐

Theorem 6.8. *Maximin-*#AV_{D}-Control *and Maximin-*#DV_{D}-Control *are* #P*-metric-complete.*

Proof. We get the result by applying the ideas from theorems 5.5 and 5.8 to maximin system. □

6.3. Summary

As we could see in this chapter, control in maximin system is computationally hard in every considered case (in case of deleting candidates we only conjecture that this result holds). Thus, each studied variant of the winner prediction problem is computationally hard in maximin. This fact is interesting also because of the fact that we proved there exists a polynomial-time algorithm that solve the decision variant of control by deleting candidates. However, it is very likely there are no polynomial-time algorithm that solves the counting variant of such control type. We leave the proof of this fact for future research.

Chapter 7

Summary

In this thesis, we have considered a natural model of predicting election winners in settings where there is uncertainty regarding the structure of the election (i.e., regarding the exact set of candidates and the exact collection of voters participating in the election). We have shown that our model corresponds to counting variants of election control problems (specifically, we have focused on election control by adding/deleting candidates and voters).

We have considered four voting rules: plurality, approval, Condorcet, and maximin voting. It turned out that the complexity of counting the number of solutions for constructive control problems under plurality, approval, and Condorcet systems is analogous to the complexity of verifying if any solution exists. That is, whenever the decision variant of the constructive problem is in P, the counting variant is in FP; whenever the decision variant is NP-complete, the counting variant is #P-complete. Only in maximin system things may be slightly different—we believe the problem regarding deleting candidates (both constructive and destructive cases) in the decision variant is computationally easy, while in the counting variant it is computationally hard.

Table 1 gathers all the results we have collected in the previous chapters. In the first two columns we can observe some symmetry, as approval and Condorcet voting have identical results, and plurality voting looks like the opposite to them—altering the set of candidates is computationally hard and altering the collection of voters is computationally easy—unlike it is for approval and Condorcet. What is interesting, maximin has #P-completeness in every considered case—every election control problem in counting variant is computationally hard under this system (however, in case of deleting candidates it is only a conjecture). It means that the winner cannot be predicted quickly under maximin voting no matter if we add or delete voters, or if we add or delete candidates (assuming Conjectures 6.4 and 6.5).

Our research can be further extended in many ways. The most natural research direction currently is to study counting variants of control under further election systems, and to provide proofs that will more accurately characterize our #P-metric-complete problems—whether or not these problems belong to a narrower class of #P-

Problem	Plurality	Approval/Condorcet	Maximin
#AC_C-Control	#P-p-complete	–	#P-p-complete
#AC_D-Control	#P-m-complete	FP	#P-m-complete
#DC_C-Control	#P-p-complete	FP	#P-p-complete (?)
#DC_D-Control	#P-m-complete	–	#P-m-complete (?)
#AV_C-Control	FP	#P-p-complete	#P-p-complete
#AV_D-Control	FP	#P-m-complete	#P-m-complete
#DV_C-Control	FP	#P-p-complete	#P-p-complete
#DV_D-Control	FP	#P-m-complete	#P-m-complete

Table 1: The complexity of counting variants of control problems. A dash in an entry means that the given system is immune to the type of control in question, #P-p-complete stands for #P-parsimonious-complete, and #P-m-complete stands for #P-metric-complete. An entry with a question mark means that the given result has not been proven, although we conjecture it holds.

completeness. One could also consider more involved probability distributions of candidates/voters that join/leave the election, as the restrictions we assumed in this work are very rigid. Another approach to winner prediction problem would be to conduct experiments and computer simulations (e.g., on some special cases), as well as to design heuristic methods and approximation algorithms to deal with #P-completeness.

Bibliography

[AKS04] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

[ASS02] K. Arrow, A. Sen, and K. Suzumura, editors. *Handbook of Social Choice and Welfare, Volume 1*. Elsevier, 2002.

[BD09] N. Betzler and B. Dorn. Towards a dichotomy of finding possible winners in elections based on scoring rules. In *Mathematical Foundations of Computer Science 2009*, volume 5734, pages 124–136. Springer-Verlag, 2009.

[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.

[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, 2003.

[CSL07] V. Conitzer, T. Sandholm, and J. Lang. When are elections with few candidates hard to manipulate? *Journal of the ACM*, 54(3):1–33, 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, 2001.

[Dod76] C. Dodgson. *A method of taking votes on more than two issues.* 1876. Pamphlet printed by the Clarendon Press, Oxford, and headed “not yet published”.

[EFS10] E. Elkind, P. Faliszewski, and A. Slinko. Cloning in elections. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, pages 768–773. AAAI Press, 2010.

[ER93] E. Ephrati and J. Rosenschein. Multi-agent planning as a dynamic search for social consensus. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 423–429. Morgan Kaufmann, 1993.

[FHH09a] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. How hard is bribery in elections? *Journal of Artificial Intelligence Research*, 35(1):485–532, July 2009.

[FHH09b] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. How hard is bribery in elections? *Journal of Artificial Intelligence Research*, 35(1):485–532, 2009.

[FHH10] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. Using complexity to protect elections. *Communications of the ACM*, 53(11):74–82, 2010.

[FHH11a] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. The complexity of manipulative attacks in nearly single-peaked electorates. *CoRR*, abs/1105.5032, 2011.

[FHH11b] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. Multimode control attacks on elections. *Journal of AI Research*, 40:305–351, 2011.

[FHHR09a] P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Llull and Copeland voting computationally resist bribery and constructive control. *Journal of Artificial Intelligence Research*, 35(1):275–341, June 2009.

[FHHR09b] 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*, pages 375–406. Springer, 2009.

[FO10] P. Faliszewski and M. Ogihara. On the autoreducibility of functions. *Theory of Computing Systems*, 46(2):222–245, 2010.

[For09] L. Fortnow. The status of the P versus NP problem. *Communications of the ACM*, 52(9):78–86, 2009.

[FP10] P. Faliszewski and A. Procaccia. AI's war on manipulation: Are we winning? *AI Magazine*, 31(4):53–64, 2010.

[FW11] P. Faliszewski and K. Wojtas. Possible winners in noisy elections. In *IJCAI Workshop on Social Choice and Artificial Intelligence*, August 2011.

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

[GJ90] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of* NP*-Completeness*. W. H. Freeman and Company, 1990.

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

[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 Caroll's 1876 voting system is complete for parallel access to NP. *Journal of the ACM*, 44(6):806–825, 1997.

[HHR07] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Anyone but him: The complexity of precluding an alternative. *Artificial Intelligence*, 171(5–6):255–285, 2007.

[HHR09] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Hybrid elections broaden complexity-theoretic resistance to control. *Mathematical Logic Quarterly*, 55(4):397–424, 2009.

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

[KL05] K. Konczak and J. Lang. Voting procedures with incomplete preferences. In *JCAI-05 Workshop on Advances in Preference Handling*, pages 124–129, 2005.

[Kre88] M. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509, 1988.

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

[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.

[Tid87] T. Tideman. Independence of clones as a criterion for voting rules. *Social Choice and Welfare*, 4(3):185–206, 1987.

[Val79] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[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, 2008.

[Zan91] V. Zankó. #P-completeness via many-one reductions. *International Journal of Foundations of Computer Science*, 2(1):76–82, 1991.