2007

# Approximation and elections

Eric Brelsford

# Approximation and Elections

Eric Brelsford
Computer Science Department
Rochester Institute of Technology

May 21, 2007

# Contents

# Chapter 1

# Introduction

Any culture that requires that a decision be made within a group necessarily creates methods for aggregating each individual's preferences. For instance, we see such a need in political elections, committees, and businesses. With the Internet and the increasing use of multiagent software systems, the general need for means of aggregating differing preferences has increased dramatically.

Voting is one way to come to a single option (or small group of options) out of a larger pool of candidate options. Many voting systems exist, and criteria exist (within the field of social choice theory) for deciding which most fairly and accurately take into account the preferences of each voter.

Since there is generally much to be gained from influencing such a vote through manipulation or bribery, one desirable criterion of fairness would be whether such activities are impossible in the system. However, it has been shown that a reasonable system that disallows manipulation does not exist [18, 43], so the next-best solution would be a system in which deciding how to bribe or otherwise influence the vote is so computationally difficult as to render it impossible or highly unlikely.

While the debate over which voting systems are most fair and effective is on record of existing over the past few centuries (and likely goes further back to ancient Greece), there may exist the seeds of a renewal of this debate in the current boom in voting due to new technologies. For one, in artificial intelligence agents may vote to determine the best course of action to take given the individual's preferences. In addition, algorithms in search engines and meta-search engines do order results in a manner that assumes a ranking was somehow approached. Voting is not only on the rise in software, of course, as most any user of the Internet could demonstrate. Internet users

routinely vote most any user of the Internet could demonstrate. Internet users routinely vote online in situations ranging from the inane (e.g., rating a video on YouTube) to the potentially crucial (e.g., voting on whether a story is newsworthy or not on any of a plethora of such sites, including Digg, Reddit, and Newsvine).

These newer uses of voting systems are interesting. They are used in environments where there are potentially far more candidates and voters than are conventionally seen in, say, political elections. Also, in these new environments, voting and manipulation can be automated to some degree, thus making the possibility of manipulation and control even more real than it has been in the past.

Faliszewski, Hemaspaandra, and Hemaspaandra have proved for a number of voting systems that the bribery problem is too complex to be feasible (i.e., NP-complete) [15], and much research has been put forth determining the complexity of other problems related to voting.

But it is still possible in the optimization cases of these problems that there exist approximation algorithms that can find a good solution with a reasonable amount of computation. That is, while a voting system may seem "resistant" to a particular form of manipulation as described by previous research, it may be that the problem is not as difficult if we allow a constant amount of error. Or, it may be that the problem is still difficult when error is allowed, thus making the voting system even more resilient with respect to some forms of manipulation. This thesis will examine the possibility of such approximations for some problems in elections.

## 1.1 Overview

We will start with some background on traditional social choice theory, including definitions of voting systems and criteria by which these systems are judged (**Chapter 2**). This background along with a small amount of computer science theory will be sufficient for the next chapter, **Chapter 3**, which will sketch the current state of the field of computational social choice. Some of the more relevant problems that are often studied under the umbrella of elections and voting systems will be defined. In addition, the known complexity of the problems will be summarized for some voting systems.

Next we will veer more towards theoretical computer science in **Chapter 4**, an exploration of approximation algorithms. This chapter contains the requisite definitions and concepts dealing with optimization problems and their approximations. It will likely be most useful as a reference when

later chapters mention an approximation concept, as opposed to being read through on its own.

Once this background is covered, the chapters containing original results begin.

First, we find some general approximation results on the problem of bribery in voting systems in **Chapter 5**. These proofs rely on the complexity of manipulation problems for the same voting systems.

Next, we attempt to show the approximability of a specific bribery problem, known as plurality-weighted-$bribery. This takes part in a few chapters.

The first chapter regarding this problem is **Chapter 6**, which deals with the knapsack problem, a classic and simply stated combinatorial problem. This chapter will consist of a number of definitions as well as an overview of some generalizations of the problem and approaches that have been taken towards solving them. The motivation behind studying the knapsack problem is the potential relation between it and the bribery problem at hand that could be exploited. The knapsack problem could also prove to be useful regarding other problems in the realm of computational social choice, and we hope that this chapter might draw some connections between the two realms that have not been previously exploited.

Given this, in **Chapter 7**, we will take a look at the approximability of the bribery problem plurality-weighted-$bribery. While we do not directly prove approximability results for this problem, we do prove that a special case of the problem is not approximable.

Finally, **Chapter 8** contains an analysis of the approximation of some control problems. For some problems we can prove that they are at least partially resistant to being approximated, while other problems are far more resistant.

## 1.2 Tips for Reading

A minimal knowledge of computational complexity theory will be assumed throughout this work. An understanding of the complexity classes P and NP as well as the concept of NP-completeness will be essential at points. Any textbook regarding algorithms or complexity theory will likely have an adequate review of these principles. Garey and Johnson's canonical book [17] can also provide this background.

Those who are already familiar with the background presented in Chapters 2 through 4 will likely be more interested in the new results presented in this thesis. These results are present in the latter chapters, specifically:

- Sections 5.2 and 5.4 contain general results regarding the approximability of bribery,
- Sections 7.2 and 7.3 contain our results pertaining to bribery in elections using the plurality voting scheme, and
- Section 8.1 holds our non-trivial proofs of the nonapproximability of control by adding candidates to plurality elections.

Some of the proofs (especially those in Chapter 8) are especially tedious. It is advisable to read the proofs in Chapter 5 first, as those follow the same pattern and are far simpler.

## 1.3 Acknowledgments

I would like to thank my advisor, Edith Hemaspaandra, for all of her guidance and suggestions. Piotr Faliszewski was also extremely helpful, namely by making a number of corrections and helping me clarify some points in this thesis.

This research was supported in part by grant NSF-CCR_0311021.

# Chapter 2

# Social Choice Theory

Social choice theory is an area of research with close ties to political science and economics that studies methods for selecting some options out of many by aggregating the preferences of a number of people or agents. This might include voting on a leader out of a number of candidates, deciding whether a proposal should become law, or settling an argument over what to make for lunch.

In particular, work in the area generally attempts to decide which voting system is most fair for a particular task. In this capacity, the beginning of social choice theory is usually dated as the late eighteenth century in France[1], when Borda and Condorcet had a disagreement over which voting system should be used in the French Academy of Sciences [42]. Both parties in the argument were mathematicians, and mathematics, including computer science, has continued to be a strong force in the field since its inception.

As software has become more complex and pervasive, concepts from social choice have progressed beyond describing methods of collective decision making in societies. Some computer systems also use these voting systems in order to decide which course of action is best, much as societies do, and this has contributed to an increased interest in the field.

## 2.1 Voting Systems

What follows is a discussion of some of the more well-known voting systems. A voting system takes a set of voters with preferences over some set of candidates and determines a set of winners. We will assume that preferences

[1] Although, recently, it has become clear that the theory has been studied as early as the thirteenth and fifteenth centuries by Llull and Curanus[20]

over candidates are strict, that is, a voter is not allowed to be indifferent between two candidates, and we will represent a voter's or a group of voters' preferences by listing them in order of most- to least-preferred. For example, a voter preferring candidate $c$ to $a$ to $b$ would be represented by $c > a > b$.

An election, $E$, is then represented as a pair of sets, $(C, V)$, where $C$ is the set of candidates and $V$ is the set of voters.

### 2.1.1 Scoring Protocols

Scoring protocols are generally straightforward: a list of points (often called a *scoring vector*, $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_m)$) to be assigned to the candidates in the election according to their position in each voter's preferences is defined by the protocol. That is, a voter has preferences $c_1 > c_2 > \ldots > c_m$, and for $1 \leq i \leq m$ assigns $\alpha_i$ points to candidate $c_i$. Those candidates with the most points win the election.

**Example 2.1.1.** *Consider a scoring protocol over* 3 *candidates with scoring vector* $(5, 4, 3)$*. If these candidates are* $\{c_1, c_2, c_3\}$ *and a voter most prefers* $c_3$*,* $c_1$ *next, and* $c_2$ *least (i.e.,* $c_3 > c_1 > c_2$*), then the voter gives* 5 *points to* $c_3$*,* 4 *to* $c_1$*, and* 3 *to* $c_2$*.*

#### plurality

Also known as simple majority or first-past-the-post, this is the most common voting system used in politics today. Part of its popularity is certainly due to its simplicity: the candidate(s) with the largest number of votes is the winner, where the candidate receives a vote whenever it is most preferred by a voter.

Plurality is a family of scoring protocols with scoring vector $(1, 0, \ldots, 0)$, that is, a voter's most preferred voter gets a point, no other candidates get any points from the voter. Due to the simplicity of the scoring vector, in practical use one rarely thinks of plurality as a scoring protocol, but rather of a vote that gives one point to the most-preferred candidate.

#### Borda count

Each voter's most-desired candidate gets $|C| - 1$ points, the second most-desired candidates get $|C| - 2$ points, etc., where $|C|$ is the number of candidates in the election. More formally, the scoring vector is $(|C| - 1, |C| - 2, \ldots, 0)$.

This method is appealing over plurality because voters are able to give their opinions of all of the candidates (as opposed to just their favorite one) and the degree of separation between candidates in a ranking is kept track of (in addition to their ordering). For a more in-depth yet very accessible discussion on the relative merits of Condorcet's (below) and Borda's counts, see [42].

**veto**

A voter gives one point to each candidate except for the one it least prefers, which gets zero points. That is, the scoring vector $(1, 1, \ldots, 1, 0)$ is applied to each voter's preferences.

### 2.1.2 Other Voting Systems

While scoring protocols are among the simplest useful voting systems to define, there are many other voting systems, some of which are used in practical settings. What follows is a mere sampling of these systems, some of which are looked at in more depth in later chapters.

**single transferable vote**

Single transferable vote (STV) takes place in a number of rounds. During each round, the candidate with the least votes (as judged in a plurality election; ties are handled differently in separate versions) is eliminated, and all voters who were voting for this candidate vote for their next preferred candidate in the next round.

**Example 2.1.2.** *Consider the election described by the following voters and their preferences:*

| *Voters* | *With Preferences* |
|---|---|
| 4 | $x > y > z$ |
| 3 | $z > y > x$ |
| 2 | $y > z > x$ |

*First, $y$, with two points, would lose to both $x$ and $z$, who get four and three points, so $y$ is eliminated. When $y$ is removed from consideration, the two voters who most prefer $y$ vote instead for $z$. The easiest way to think about this might be that when a candidate loses a round, it is entirely removed from each voter's preferences. Thus, $z$ ends up with five points and beats $x$, which still only has four points.*

*If the election has more candidates, the process is identical, but there are more rounds where candidates are eliminated until the desired number of winners is found.*

Single transferable vote is used in some regions from the local to federal levels, perhaps most notably in Australia.

### Condorcet voting

Under Condorcet voting, the candidate who strictly beats each other candidate in a pairwise contest wins overall.

**Example 2.1.3.** *Say we had the following election:*

| *Voters* | *With Preferences* |
|---|---|
| 3 | $x > y > z$ |
| 2 | $z > y > x$ |
| 2 | $y > z > x$ |

*then, overall, the voters prefer $y$ to $x$ (four points to three), $z$ to $x$ (also four points to three), and $y$ to $z$ (five points to two). Since $y$ beats both $x$ and $z$ (all of the other candidates) when it goes up against them individually, $y$ would be the Condorcet winner.*

Such a criterion for a fair election seems a common-sense choice, however, there will not always be a Condorcet winner due to cycles in preferences. This is often called Condorcet's Paradox.

**Example 2.1.4.** *To see how the paradox appears, consider the following election over three candidates, $x$, $y$, and $z$:*

| *Voters* | *With Preferences* |
|---|---|
| 3 | $x > y > z$ |
| 3 | $y > z > x$ |
| 3 | $z > x > y$ |

*There is no Condorcet winner in the above election. To see this, we first compare candidates $x$ and $y$, and we see that six voters prefer $x$ and three voters would rather $y$. Then, when comparing $y$ and $z$, six voters prefer $y$ and three $z$. So far, it looks as though $x$ should be the winner of the election since it beat $y$ between the two and $y$ beat $z$. However, we have yet to compare $x$ and $z$, and, when we do, we are chagrined to find that six voters prefer $z$ and three $x$. That is, the group of voters has decided that its preferences are $x > y > z > x$, which is irrational and leaves us with no better of an idea of who should win the election.*

### Dodgson voting

Dodgson voting is an attempt to sidestep the Condorcet paradox by making the candidate who is closest to being a Condorcet winner the winner. Determining which candidate is the closest consists of counting the number of switches that would need to be made in the voters' preferences in order to make the candidate a Condorcet winner. Then, the candidates who have the smallest value for this number is the Dodgson winner.

A "switch" in this context is changing a voter's preferences by flipping two adjacent candidates in the preferences. For example, changing a voter's preferences from $z > x > w > y$ to $z > x > y > w$ would count as one switch.

This voting system is named after its creator, the mathematician Charles Dodgson, who is better known by his pseudonym Lewis Carroll.

### approval voting

In this system, each voter gives a point to each candidate he or she approves of, and a candidate with the highest number of points wins.

**Example 2.1.5.** *There might be an election over candidates* $(v, w, x, y, z)$ *with the following voters:*

| *Voters* | *With Preferences* |
|---|---|
| 2 | $(1, 0, 0, 1, 1)$ |
| 2 | $(1, 1, 1, 1, 0)$ |
| 2 | $(0, 1, 0, 0, 1)$ |
| 1 | $(0, 0, 1, 0, 1)$ |

*That is, there are two voter who vote with preferences* $(1, 0, 0, 1, 1)$*, meaning that they approve of candidates* $v$*,* $y$*, and* $z$*, but not* $w$ *or* $x$*. Here* $z$ *wins with five points—five voters approve of candidate* $Z$*.*

At times, $k$-approval voting, where a voter must approve of exactly $k$ candidates, or other situations, such as when $k$ is half the number of candidates, are examined.

### maximin

This system assigns a score to candidate $x$ by, in effect, holding plurality elections with each other candidate in the election. The score of $x$ is the lowest score that it gets in any of these elections. The highest score is clearly most desirable and determines the winner.

## 2.2 Criteria in Social Choice Theory

Much research has been devoted to finding voting systems which are fair. Generally, we consider a fair election to be one in which the will of the voters is reasonably expressed. There are a number of criteria that a fair election scheme could consist of, and here are a few concrete ones:

#### Pareto condition

A voting system satisfies the Pareto condition if, when candidate $x$ is preferred by all voters over candidate $y$, candidate $x$ will be preferred over candidate $y$ overall, as measured by the voting system. One modification of this condition is to make it weak by saying that $x$ must be strictly preferred to $y$ in each voter's preferences (i.e., no indifference is allowed in preferences between $x$ and $y$). Similarly, the condition can be made strong by stipulating that only one voter need strictly prefer $x$ to $y$, while the rest may find $x$ at least as good as $y$.

#### monotonicity

If a voting system never ranks a candidate lower because some voters rank the candidate higher than before, the voting system is said to obey monotonicity.

Many voting systems do obey this criterion. The voting systems that are not monotone are often those which take place in rounds. One such voting system is single transferable vote.

#### independence of irrelevant alternatives

This condition (in [2]) states that if candidate $x$ is preferred over candidate $y$ overall, and one or more voters change their preferences regarding candidates other than $x$ and $y$, $x$ should still be preferred to $y$ overall.

While this condition seems to be a desirable one, many voting systems do not obey it.

**Example 2.2.1.** *Consider a Borda election like this:*

| *Voters* | *With Preferences* |
|---|---|
| 2 | $w > x > y > z$ |
| 3 | $w > z > x > y$ |

*Clearly, the voters prefer $x$ over $z$ with scores 7 and 6, respectively. However, say we modify the preferences of both of the voters from the first set from $w > x > y > z$ to $w > y > x > z$. Now, even though the voters changed their minds by promoting $y$, the overall preferences now favor $z$ over $x$ with scores 6 and 5. Although $y$ is not relevant to the relative rankings of $x$ and $z$, the ranking of $y$ does have an effect on which of the two is preferred overall.*

**non-dictatorial**

We should certainly expect a voting system to not give one voter the ability to sway the outcome of the election. Otherwise, if a single voter ultimately decides the outcome of an election, the scheme is *dictatorial*. That is, if some voter, $d$, most prefers candidate $x$ in an election and $x$ wins no matter what the rest of the voters prefer, then the voting system being used is dictatorial, but if there is no such $d$, then the system is non-dictatorial.

## 2.3 Impossibility

Some of the work in social choice theory more relevant to this thesis includes impossibility results, that is, proofs that particular properties cannot exist, or that certain sets of properties cannot coexist, in voting systems.

### 2.3.1 Arrow's Theorem

This theorem, which was discovered by the economist and Nobel Prize winner Kenneth Arrow, attempts to determine whether a "fair" voting system can exist. It states that a voting system over three or more candidates cannot at the same time

- obey monotonicity,
- obey independence of irrelevant alternatives,
- ensure that any possibility out of the candidates could be selected, and
- be non-dictatorial,

all of which seem to be reasonable criteria for a "fair" voting system.

While the theorem does not necessarily imply that all voting systems are worthless and broken, it does suggest that one of the above criteria will have to be weakened for any non-dictatorial system.

One formulation and proof of this is in [2], though it has since been made stronger (e.g., by weakening some of the above conditions) and its proof has been simplified [40].

### 2.3.2 Gibbard-Satterthwaite Theorem

This theorem, found independently by Gibbard and Satterthwaite, is similar to Arrow's. It states [18, 43] that a voting system over three or more preferences cannot at the same time

- ensure that any possibility out of the candidates could be selected,
- be immune to manipulation, and
- be non-dictatorial.

Manipulation in the context of voting systems occurs when a voter or set of voters are better off not voting with their true preferences and, so, choose to vote strategically in order to gain a better outcome. That is, the manipulating voters will see a more preferred candidate win by voting with preferences that differ from their actual ones. Certainly it is desirable to use a voting system that is not manipulable, since a voting system does not feel fair or effective if it does not make voters feel comfortable with expressing their true preferences because other voters might be manipulating the election.

In [40], Reny looks at the relationship between Arrow's and the Gibbard-Satterthwaite theorem in more detail. In fact, the author uses what is essentially the same, relatively simple proof to show both theorems hold.

As this theorem shows that manipulation will *always* be a possibility when more than three candidates are involved in a reasonable voting system, it motivates the current research into the computational complexity of performing this manipulation and of computational social choice in general. Such research can give us a better idea of *how* susceptible a system is to manipulation (since we know that essentially all are susceptible) or some related problem and in some ways this research can tell us which systems are better than others.

With that, let us now turn to computational social choice, a field that utilizes computational complexity theory to better determine vulnerabilities which are exposed in social choice theory.

# Chapter 3

# Computational Social Choice

Whereas social choice theory is interested in the existence of voting systems with particular properties (Chapter 2), computational politics, or computational social choice, is an attempt to determine the computational difficulty of problems dealing with particular aspects of voting systems.

Similarly to social choice theory, a good deal of focus has been spent on finding the complexity of performing possibly mischievous actions on an election in a given voting system. The key difference is that computational social choice can in some instances reveal that, while, yes, this form of manipulation is *possible* in such-and-such voting system, finding the way to do so is NP-complete and therefore can generally be considered intractable. In this way, computational social choice can give us a better characterized picture of which voting systems are vulnerable to certain bad occurrences.

In addition to these somewhat negative problems, effort has been put into determining the complexity of certain positive problems, such as determining the winner of an election or the scores of candidates in an election. Perhaps surprisingly, for some voting systems these problems are rather difficult. In fact, some are unlikely to be in NP[1].

## 3.1 Definitions and Notation

Before we get into the definitions of the various problems to be considered, we should define the notation to be used.

An election is a tuple $(C, V)$ over a set $C$ of candidates and a multiset $V$ of voters. Each $v \in V$ has a list of preferences over all candidates in $C$,

[1]See, for example [22], where winner determination for Dodgson voting is shown to be complete for parallel access to NP

*prefs* $= c_1 > c_2 > \ldots > c_n$ where $c_1$ is more preferred by $v$ to $c_2$, etc. If an election is *weighted*, each $v$ must also have a non-negative integer weight, $\omega$.

Given an election $(C, V)$ and a candidate $c \in C$, we will denote $c$'s score in the election (if the election is taking place in a voting system that assigns scores to candidates, as will always be the case in this thesis) as $score_{(C,V)}(c)$, and the value of this score is as defined by the voting system at hand, which should be clear from context. In many cases, depending on the voting system, a *winner* of $(C, V)$ is any $c \in C$ such that $score_{(C,V)}(c)$ is the largest score in $(C, V)$. As a set of candidates can win an election, we will sometimes look for a *unique* winner, one which is the sole winner of an election.

### 3.1.1 Instance Encoding

For any problem in this area, we will assume that the instances are encoded reasonably and that each voter and its preferences are an entry in the instance, much like a ballot in an election. In some cases, we will instead look at *succinct* elections, where one entry may represent multiple, identical voters.

For instance, if there are $k$ voters with identical preferences (and weights, if weights may vary), then each must be separately represented in a non-succinct election, that is, there are $k$ distinct entries in the election. On the other hand, the same $k$ voters would be represented in a succinct election as a single entry, but with the frequency count $k$.

While this may seem to be a small difference, succinctness can have an impact on the complexity of problems dealing with elections. This is so because complexity is measured as a function of the length of the input, and input length for succinct elections can be significantly smaller than in the non-succinct case. We will encounter this in some detail in Section 7.2.

## 3.2 Control

The study of control in elections deals with the possibility of affecting the outcome of elections by changing the structure or makeup of the election. This involves a chairperson in charge of the election adding or removing voters or candidates, or partitioning voters or candidates into subelections that precede the main election.

When attempting to exert control, one might go one of two directions. One could try to make a particular candidate become the winner of the election. This is called *constructive* control. On the other hand, *destructive*

control is used when trying to make a certain candidate lose an election. In either case, traditionally in control a candidate wins an election only if it is the *unique winner*, that is, there are no ties allowed. Therefore, when we are trying to make a particular candidate lose the election, it is sufficient to make some other candidate tie the given candidate in order to make it lose.

The complexity of a number of control problems have been studied in some depth. In [5], Bartholdi, Tovey, and Trick initiate the research of control by looking at the complexity and possibility of constructively controlling elections under the plurality and Condorcet voting schemes.

The destructive case was studied by Hemaspaandra, Hemaspaandra, and Rothe in [21], for the plurality, Condorcet, and approval voting. The authors also performed the analysis for approval in the constructive cases, since [5] did not deal with approval.

Please refer to Chapter 2 for background on the voting systems mentioned. As the definitions and results of the aforementioned papers will be used in Chapter 8, let us present them here.

### 3.2.1 Definitions

Control problems largely include adding or deleting candidates or voters or partitioning candidates or voters in order to make a particular candidate win or lose the election. The problems are defined identically for each voting scheme (with the exception that in the case of approval voting, a voter's list of preferences is a 0-1 vector the size of the set of all potential candidates as opposed to an ordered list of candidates), and the constructive versus destructive cases are similar enough that we can collapse them into the same definition. The constructive case of each question will be stated with the modifications for the destructive case in brackets. Nothing changes between the two cases within the "Given" field of the problems. The following definitions are adapted from [5, 21].

#### Adding Candidates

**Given:** A set $C$ of qualified candidates and a distinguished candidate $c \in C$, a set $D$ of possible spoiler candidates, and a set $V$ of voters with preferences over $C \cup D$.

**Question:** Is there some subset of $D$ whose entry into the election would assure that $c$ is [not] the unique winner?

In this case, the voters' preference lists are assumed to be over all potential candidates. That is, all candidates who might be up for election, including those in $D$, are considered.

### Deleting Candidates

**Given:** A set $C$ of candidates, a distinguished candidate $c \in C$, a set $V$ of voters, and $k \in \mathbb{Z}^+$ with $k \leq |C|$.

**Question:** Is there some subset of $k$ or fewer candidates in $C$ $[C - \{c\}]$ whose disqualification would assure that $c$ is [not] the unique winner?

### Partitioning Candidates

When partitioning candidates, there are two intuitive ways that candidates from either segment of the partition might compete:

- winners are determined from within the first segment defined by the partition that go on to compete with all of the candidates in the second segment defined by the partition, or
- winners from both segments are found and compete in a run-off election.

In either case, an instance of the problem is the same:

**Given:** A set $C$ of candidates, a distinguished candidate $c \in C$, and a set $V$ of voters.

The question for the first case, *Control by Partition of Candidates* is:

**Question:** Is there a partition of $C$ into $C_1$ and $C_2$ such that $c$ is the unique winner in the sequential two-stage election in which the winners in the subelection $(C_1, V)$ who survive the tie-handling rule move forward to face the candidates in $C_2$ (also with the voter set $V$)?

The question for the second case, *Control by Run-Off Partition of Candidates* is:

**Question:** Is there a partition of $C$ into $C_1$ and $C_2$ such that $c$ is the unique winner of the election in which those candidates surviving (with respect to the tie-handling rule) subelections $(C_1, V)$ and $(C_2, V)$ have a run-off with voter set $V$?

In any partition, as described by [21], there are two tie-handling rules:

- *TE*, "ties eliminate," where candidates that tie during subelections do not move on to the next round, and
- *TP*, "ties promote," where candidates that tie during subelections do move on to the next round.

As is clear in Table 3.1, tie-handling rules do sometimes affect the complexity of the partition control, depending on the voting system.

**Adding Voters**

**Given:** A set $C$ of candidates, a distinguished candidate $c \in C$, a set $V$ of registered voters, a set $W$ of unregistered voters, and $k \in \mathbb{Z}^+$ with $k \leq |W|$. All voters have preferences over the entire set $C$.

**Question:** Is there a set of $k$ or fewer voters in $W$ whose registration would assure that $c$ is [not] the unique winner?

**Deleting Voters**

**Given:** A set $C$ of candidates, a distinguished candidate $c \in C$, and $k \in \mathbb{Z}^+$ with $k \leq |V|$.

**Question:** Is there a set of $k$ or fewer voters in $V$ whose deletion would assure that $c$ is [not] the unique winner?

**Partitioning Voters**

**Given:** A set $C$ of candidates, a distinguished candidate $c \in C$, and a set $V$ of voters.

**Question:** Is there a partition of $V$ into $V_1$ and $V_2$ such that $c$ is the unique winner in the hierarchical two-stage election in which the survivors of $(C, V_1)$ and $(C, V_2)$ run against each other with voter set $V$?

As in the candidatepartitioning problems, [21] contains an examination of the voterpartitioning problem in both the case where ties eliminate candidates ($TE$) from the run-off election and the case where ties promote candidates ($TP$) to the next round. Again, the tie-handling rules do have an effect on some voting system's vulnerability to this flavor of control.

| | Plurality | | Condorcet | | Approval | |
|---|---|---|---|---|---|---|
| Control by | Construct. | Destruct. | Construct. | Destruct. | Construct. | Destruct. |
| Adding Candidates | R | R | I | V | I | V |
| Deleting Candidates | R | R | V | I | V | I |
| Partition of Candidates | TE: R<br>TP: R | TE: R<br>TP: R | V | I | TE: V<br>TP: I | TE: I<br>TP: I |
| Run-off Partition of Candidates | TE: R<br>TP: R | TE: R<br>TP: R | V | I | TE: V<br>TP: I | TE: I<br>TP: I |
| Adding Voters | V | V | R | V | R | V |
| Deleting Voters | V | V | R | V | R | V |
| Partition of Voters | TE: V<br>TP: R | TE: V<br>TP: R | R | V | TE: R<br>TP: R | TE: V<br>TP: V |

Table 3.1: ([21]): Summary of results regarding the complexity of control problems for plurality, Condorcet and approval voting schemes as presented in [21]. Constructive results for plurality (except for TE and TP for Partition of Voters) and Condorcet via [5], the rest are from [21]. I = immune, R = resistant, V = vulnerable, TE = Ties-Eliminate, TP = Ties-Promote.

### 3.2.2 Results

Table 3.1 presents the results regarding the computational complexity of the voting systems Plurality, Condorcet, and Approval. The results label each system as *immune*, *resistant*, or *vulnerable* to the given form of control. If a system is immune to a particular type of control, it is impossible to perform that control on an election in the system—that is, the preferred outcome cannot be attained by performing the respective control technique. If such control is possible, in these cases computing it is NP-hard (resistant) or in P (vulnerable).

Since the immune and vulnerable control problems are in P and are thus easily optimizable, we will be more interested in those which are proven resistant. In Chapter 8 we examine the approximability of those control problems which are resistant.

## 3.3 Manipulation

Manipulation is generally the problem of voters voting with dishonest preferences for their own benefit. Like control, this problem has been studied under numerous voting systems, in both constructive and destructive situations, and with a few variations, such as where the voters are weighted.

### 3.3.1 Definition

Generally, these problems are defined as follows:

**$\mathcal{E}$-manipulation**

**Given:** A set $C$ of candidates, a set $V$ of voters, a set $S$ of potential manipulative voters such that $V \cap S = \emptyset$, and a distinguished candidate, $p$.

**Question:** Is there a setting of the preference lists of the voters in $S$ such that $p$ is *a* winner of election $(C, V \cup S)$ under voting system $\mathcal{E}$?

Note that, in contrast to the above definitions of control problems, we here attempt to make $p$ a winner, but not necessarily the sole winner, of the election.

This is for the constructive case, where the manipulators try to make $p$ win the election. In the destructive case, the problem is defined identically, with the difference that the manipulators attempt to make $p$ lose. For the weighted cases, everything is exactly the same except, of course, the voters have weights.

### 3.3.2 Results

The study of the complexity of manipulation was initialized for the unweighted case was determined for some voting systems by Bartholdi, Tovey, and Trick [26].

In [13], Conitzer, Lang, and Sandholm look at how many candidates it takes for manipulation to become NP-complete for various voting systems in the weighted case, and, in doing so, produce a more complete picture of the complexity of manipulation. In the constructive case, manipulation becomes NP-complete for many systems rather quickly if the votes are weighted, as shown in Table 3.2. In the destructive case, the problem remains in P for many of the systems analyzed, though they did find that it was NP-complete for STV and plurality with a runoff.

Conitzer and Sandholm, in [14], try to find a generic method of making manipulation difficult for all voting systems. This method includes adding an elimination round to the election which precedes the actual election. The candidates are scheduled for this preround in a variety of ways, including deterministically (in which the schedule is decided before the votes are gathered), randomly (schedule decided after votes are elicited), and in an interleaved manner (in which the scheduling is done while the votes are elicited). This relatively simple set of tweaks can make a voting system PSPACE-hard in the interleaved case.

| Number of candidates | 2 | 3 | $\geq 4$ |
|---|---|---|---|
| Borda | P | NP-complete | NP-complete |
| Veto | P | NP-complete | NP-complete |
| STV | P | NP-complete | NP-complete |
| plurality with runoff | P | NP-complete | NP-complete |
| maximin | P | P | NP-complete |
| plurality | P | P | P |

Table 3.2: ([13]): The computational complexity of weighted manipulation for relevant voting systems with the given number of candidates (2, 3, or $\geq 4$) in the election. Excerpted from results originally presented in [13].

In [23] Hemaspaandra and Hemaspaandra present some dichotomy theorems regarding the complexity of manipulation for scoring protocols. We will restate the main result of this paper here for the reader's convenience (modified only to keep notation consistent).[23]

**Theorem 3.3.1** ([23], Theorem 2.1)**.** *For each $m$ and each scoring protocol $\alpha = (\alpha_1, \cdots, \alpha_m)$, $\alpha$-weighted-manipulation is in* P *if $\alpha_2 = \alpha_3 = \cdots = \alpha_m$, and is* NP*-complete otherwise.*

### 3.3.3 Approximability

The optimization of manipulation is tricky to define. Clearly, we want to minimize the number of voters that we manipulate in order to make the preferred candidate win the election.

Do we say that when optimizing manipulation we only leave in the election those voters for whom we have changed the preference lists? In this case, the voters left out are also acting in a manipulative fashion by abstaining. That is, there is little gained by the manipulators, who presumably would like to minimize their chances of being detected, since they all would be doing something manipulative.

On the other hand, do we keep all potentially manipulative voters in the election after changing some of the voters' preference lists? This would be a vast departure from the original definition of manipulation since preference lists would have to be defined for all potentially manipulative voters *a priori*. In addition to being a radical change, such a move would make optimal manipulation indistinguishable from optimal bribery (with unit prices) and, thus, would be redundant.

As we have been unable to find a suitable definition of optimal manipulation, we will not examine the approximability of manipulation, which by its nature demands an optimization problem.

## 3.4 Bribery

Bribery is a natural extension of manipulation and a natural problem in the realm of voting systems.

### 3.4.1 Definitions

**$\mathcal{E}$-bribery**

**Given:** A set $C$ of candidates, a set $V$ of voters, a distinguished candidate $c$, and a positive integer $k$, the budget.

**Question:** Is there a set of at most $k$ voters whose preferences can be changed to make $c$ the winner of election $(C, V)$ under system $\mathcal{E}$?

This is the simplest version of bribery, where each voter's price and weight are the unit price and weight, assumed to be 1. It is also interesting to look at problems where the weights ($\omega \in \mathbb{Z}^+$) vary by voter, the prices ($\pi \in \mathbb{N}$) of the voters vary, or both vary independently. These problems are referred to as $\mathcal{E}$-weighted-bribery, $\mathcal{E}$-$bribery, and $\mathcal{E}$-weighted-$bribery, respectively.

### 3.4.2 Results

The study of bribery from a computational complexity point of view was initiated in [15], by Faliszewski, Hemaspaandra, and Hemaspaandra. They find the complexity of bribery for each of the variations mentioned above, and for a number of voting systems. One interesting result of this examination is that plurality voting with weighted votes and varying prices is NP-complete for as few as two candidates (the approximability of which is studied in Chapter 7), whereas bribery in plurality voting with only one of those qualifications is in P. However, for the case where the votes are weighted and the prices vary, if either the weights or prices are encoded in unary, the problem comes back down to P.

Faliszewski, Hemaspaandra, and Hemaspaandra also present a number of duality theorems regarding the complexity of bribery in scoring protocols.

Finally, [15] also provides a look at the relationship between bribery and manipulation, the latter of which has been studied previously and rather thoroughly. In the authors' words, "bribery can be viewed as manipulation where the set of manipulators is not fixed in advance and where deciding who to manipulate is a part of the challenge." Of course, part of this challenge is that each voter might have weights or prices that vary. In addition, it is shown that any manipulation problem is many-one reducible to the equivalent $bribery problem, so some of the known manipulation results immediately give hardness results for equivalent bribery problems.

### 3.4.3 Approximability

Since these bribery problems are quite new, approximation in this context has not yet been studied. In Chapter 5 we initiate this study and look at the possibility of approximating bribery in general. Then, in Chapter 7, we examine the approximability of bribery in the plurality system when both weights and prices may vary.

## 3.5 Other Problems

Other manipulative problems, such as modifications of the above problems and the complexity of lobbying [11] have also been studied. Analyzing the above problems for other voting systems that have not yet received this treatment is also a common source of new research (e.g., [37]).

Some problems are more inherent to the voting systems themselves in that they do not assume some potential wrong-doing on the part of voters or a chairman in the election. These include finding the *score* of a candidate or the *winner* of an election in a voting system such as Dodgson's [4], which is shown to be complete for parallel access to NP [22]. There does exist an algorithm which can often find the Dodgson winner of an election [24].

## 3.6 Average-Case Elections

When creating algorithms and heuristics that deal with performing some operation on an election, it can at times be essential to be able to generate "typical" elections on which to test said software. In particular, a heuristic should be tested against elections that contain realistic voters and candidates, as otherwise it would be difficult to measure the efficacy of the heuristic.

While the most useful test cases would be data from actual elections, this can be difficult to attain for a few reasons. First, we are not aware of any countries that record the *full* preference lists of voters. Even in countries such as Australia, where the full preference lists must be gathered to hold single transferable vote elections, it appears that the only available data lists the number of voters who vote for each candidate during each round. Second, we could not possibly have the prices that voters would be willing to accept as bribes, obviously because this has not been recorded. We might also wonder if such data would be reliable, if collected, since bribery could be an emotional and irrational activity for voters, so they might charge a different amount than they might claim in response to such a poll, or they might charge more or less depending on how their votes are to be changed (which is mentioned in [15]).

### 3.6.1 Voter Preferences

The task of creating typical voter preferences has been studied both on the computational and non-computational sides of social choice theory.

In a paper from the computational perspective [12], Conitzer looks at a means of generating election instances by having each candidate and voter choose a position on some number of issues, where each position is a real value in the range $[0, 1]$. The voters then rank the candidates by their agreement on the set of issues. This is essentially the spatial model which has been studied in traditional social choice theory.

Aikaterini, in a thesis [1], mentions using essentially this spatial model, specifically for comparing Greek electoral systems. The author compares this method, using a normal distribution, to another method using a uniform distribution.

In [10], the authors look in more depth at the spatial model. Perhaps looking at this paper could, at the very least, lead to some improvements over Conitzer's approach. As the authors say, "Despite years of intensive research on elections, we remain virtually without clues as to the likely character of the probability density of elections occurring in any electoral situation with more than two candidates." This paper looks at a few parameters used when generating an election within the spatial model, including having the candidates' stances selected as randomly as the voters' are and making the candidates more and less similar to each other. These three cases illustrate the situations where a candidate is basically a voter randomly chosen, where candidates are rewarded for conceding toward the center, and where candidates are chosen based on their extremism, respectively. One result of

this analysis shows that experiments using the uniform distribution (where the number of voters having a particular preference list is nearly equal for each possible list) rather than the spatial model underestimated the usual percentage of elections that will have Condorcet winners.

Some [39] have suggested interesting additions to the spatial model of voter preferences, including direction (support versus opposition of a given issue as opposed to simply "how close is this voter to the issue?"). The result of these additions is a directional model of voter preferences. The paper includes some historical data from previous elections which could potentially be of use.

### 3.6.2 Prices and Weights

When we are considering generating elections where the voters have prices and weights that both vary independently, it makes sense to think about generating these properties the way we would generate items for the knapsack problem (see Chapter 6 for the definition and a discussion of this problem), and we do think that this is one aspect of computational social choice that could benefit from the research done on these problems.

Pisinger's thesis [36] is an in-depth study of algorithms for the family of knapsack problems. It contains a discussion of the techniques used in finding a distribution of weights versus prices. These include random, weakly correlated, and strongly correlated distributions. At least a weak correlation would likely be suitable for generating such elections since one would imagine that a voter with more weight will use this influence to gain a higher price for his or her vote.

# Chapter 4

# Approximation Algorithms

Approximation algorithms are useful for optimization problems that are believed to be intractable. In such cases, rather than having no solution at all or waiting eons for one, it is often preferable to have a procedure by which a solution within some error bound of the optimal is found in a reasonable amount of time. An approximation algorithm can then be assessed by the amount of error it allows as well as its time and space complexities.

## 4.1 NP Optimization

In the realm of decision problems, a binary, yes/no answer is all that is required. For instance, in computational social choice (Chapter 3), the problem of deciding whether a set of bribes can make a candidate win within a set budget is a decision problem. The foundation of computational complexity is on such problems (more specifically, on the languages that represent the problems), and this rigidity and universality of a binary answer lends a great deal of power to the field.

However, there are often times when one would like to know the optimal solution to a problem. For instance, in the example above, it could be most useful for a potential perpetrator of a bribe to find the cheapest way to complete a bribe, rather than seeing whether a specific amount of money will be adequate, as is the case for the decision problem. When a problem specifies that a quantity should be maximized or minimized rather than seeking a 'yes' or 'no' answer, we call it an optimization problem.

NPO is the class of optimization problems that have NP decision problems at their core, where the goal is to either maximize or minimize some quantity in the problem. For instance, the optimal bribery problem men-

tioned above clearly has the related decision problem at the heart of it. A more formal definition is given, e.g., in [6].

A subclass of NPO is PO, the set of optimization problems that can be optimized in polynomial time. A problem in PO necessarily has an underlying decision problem that is in P since solving the optimization problem simultaneously solves any instance of the decision problem. As such, the set NPO − PO contains those optimization problems whose underlying decision problems are in NP − P, or, often, are NP-complete. Of course, NPO $\neq$ PO, assuming NP $\neq$ P.

## 4.2 Approximation

As stated in the previous section, we know that difficult decision problems will have similarly difficult optimization problems, but we have more options when solving optimization problems than we do with decision problems. The key here is that we can exploit the, perhaps, *multi-valued* nature of problems in NPO, the fact that we are looking for a biggest or smallest value as opposed to the more *binary* nature of decision problems, where only 'yes' or 'no' answers are appropriate.

What if we could find an approximate solution to an optimization problem, within some error bound? That is, what if, for our optimal bribery problem (assuming that its underlying decision problem is NP-complete), we found an algorithm that could, in polynomial time, find a solution that is provably no more than, say, 10% bigger than the optimal solution? Such an algorithm could be very useful, indeed, especially considering the difficulty of finding an optimal solution.

We will refer to an *approximation algorithm* as an algorithm that, in time polynomial with respect to the length of the input, outputs a solution that differs from the optimal solution by an error bounded by some constant factor (say, twice the optimal solution).

The way in which the error bound is expressed varies a great deal in the literature. We will adopt the notation of [19] and, to a lesser degree, that of [44].

### 4.2.1 Notation

Let $z^A$ be the approximate solution output by an approximation algorithm, $A$, and $z^*$ be the optimal solution for the same input. If the problem $A$ attempts to solve is a maximization problem, then it is said to be an $\epsilon$-

approximation algorithm if

$$z^A \geq (1 - \epsilon) \cdot z^*$$

on all inputs. When $A$ is approximating a minimization problem, it is an $\epsilon$-approximation algorithm if, on all inputs,

$$z^A \leq (1 + \epsilon) \cdot z^*.$$

These inequalities stem from the definition of the *relative error* of the algorithm, $A$, as:

$$\frac{|z^* - z^A|}{z^*}.$$

Then, we can use $\epsilon$ as the error bound:

$$\frac{|z^* - z^A|}{z^*} \leq \epsilon$$

for all possible inputs, where $\epsilon \in \mathbb{R}$ and $\epsilon > 0$, and simple algebra gets us the earlier inequalities. In either the minimization or the maximization case, the closer $\epsilon$ is to zero, the closer the solution output by an approximation algorithm will be guaranteed to get to the optimal value, since an error bound of 0 would imply that the algorithm would return the optimal value.

Also note that our definitions only make sense when $z^*$ is strictly positive. That is, if $z^* = 0$, then $z^A$, no matter the error bound, is required to also be 0, the optimal solution. Therefore, following previous research, we will restrict our attention to approximating optimization problems where 0 is not a possible optimal value.

### 4.2.2 APX

APX is the subset of NPO such that the problems included within it are all approximable within a bounded error in polynomial time on the length of the input. That is, each problem that has an approximation algorithm as defined above is a member of APX.

**Example 4.2.1.** *Consider the greedy approximation algorithm for the knapsack optimization problem (Chapter 6). The problem, in short, is to pack as much profit as possible into a sack without exceeding the sack's capacity.*

*The algorithm roughly works by sorting the items that can be placed in the sack in descending order by their efficiencies (profit-to-weight ratios) and placing the items in the sack in this order if they fit.*

*Since the algorithm is a* $1/2$*-approximation algorithm*[1] *and it is for a maximization problem, we are assured that the solution output by it will be at least* $1-\epsilon = 1-1/2 = 1/2$ *of the optimal solution. That is, if the optimal solution to a problem is* 200*, the lowest approximate solution that could be output is* 100*.*

Within APX are the classes PTAS and FPTAS.

### 4.2.3 PTAS and FPTAS

For some approximation algorithms, specifically, some greedy algorithms, $\epsilon$ is fixed. While this may do for some problems or applications, it is generally useful (and more interesting) to be able to make this performance ratio as small as one needs for a particular application, although always with a cost in time or space efficiency. In this case, $\epsilon$ is provided as a parameter to the approximation algorithm. Such algorithms are often referred to as approximation *schemes* because, in effect, there is a series of distinct algorithms within the scheme, and the appropriate one is selected by the given $\epsilon$.

When $\epsilon$ is a parameter to the approximation algorithm, it can also be considered when measuring the time complexity of the algorithm.

If the time complexity of an approximation algorithm is polynomial on the length of the input, then the algorithm is called a *polynomial-time approximation scheme* (PTAS).

If a polynomial-time approximation scheme's time complexity is also polynomial on $\epsilon^{-1}$ (inverted since a lower $\epsilon$ leads to a solution closer to the optimal and thus takes more work), it is labeled a *fully polynomial-time approximation scheme* (FPTAS).

A problem in NPO with a PTAS is said to be in PTAS, and similarly a problem with an FPTAS is in FPTAS. It is easy to see that PO $\subseteq$ FPTAS $\subseteq$ PTAS $\subseteq$ APX $\subseteq$ NPO (illustrated in Figure 4.1), and these inclusions are all strict, assuming P $\neq$ NP.

The error bound, $\epsilon$, passed to a PTAS or FPTAS will always be in terms of the relative error of the algorithm as opposed to the following alternate measures of approximation.

**Example 4.2.2.** *If* $\epsilon = 0.1$ *is passed to a PTAS for a minimization problem (such as one in Section 6.4.1),*

$$z^A \leq (1+\epsilon)\cdot z^* \leq 1.1 \cdot z^*,$$

[1] We will not prove this here. See Section 6.4.1 for more.

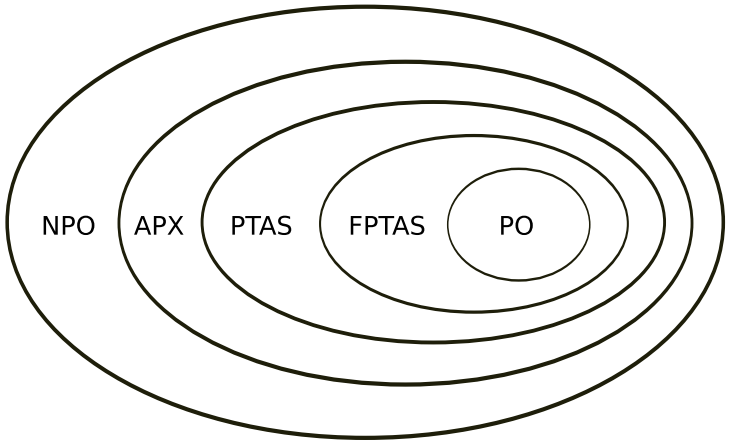


Figure 4.1: Anatomy of NPO

*so we are assured that the approximate solution returned by the PTAS will be no greater than* 110% *of the optimal solution for the instance. We also know that the time the algorithm takes to complete will be polynomial on the length of the input instance. However, we have no assurance that the runtime will grow polynomially as we get a more accurate solution,* $\epsilon < 0.1$.

*If we had an FPTAS for the same problem, we would be certain that the runtime would grow at a polynomial rate on both* $1/\epsilon$ *and the length of the input.*

### 4.2.4 Other Measures of Approximation

It is not unusual to see error bounds for approximation algorithms expressed in another way, namely, with respect to the *absolute performance ratio*

$$\frac{z^A}{z^*} \leq \alpha$$

rather than the relative error [17]. This is most common when an algorithm has a fixed value for the error—that is, when the algorithm is neither a PTAS nor an FPTAS.

In this notation, it is often the case that one will cite a "2-approximation algorithm" for a minimization problem. This can be confusing since in notation using the relative error of the approximate solution, this would mean that $z^A$ is no greater than three times greater than $z^*$, but what is really intended is that $z^A$ is, at most, twice $z^*$.

## 4.3 Some Results Regarding Approximation

A great deal of effort in computational complexity has been put into more fully describing the class of problems that are approximable. In what follows, we will attempt to summarize the main results that have been attained in the study of approximation.

### 4.3.1 Strong NP-hardness and Pseudo-Polynomial Time

First of all, there are some conditions under which it is proven that finding an FPTAS for a problem is impossible (assuming P $\neq$ NP). In order to get closer to understanding these conditions, it will be best to look first at a subset of the decision problems referred to as NP-hard.

In [16], Garey and Johnson introduce the concept of *strong* NP-hardness. If an NP-hard problem is NP-hard even when the values of the input numbers to the problem are bound by a polynomial, the problem is strongly NP-hard.

**Example 4.3.1.** *The most straightforward example of such a problem is the well-known traveling salesman problem (TSP), in which the goal is to find a Hamiltonian path in a given graph (a path through the graph that touches each node without repeating any nodes) where the sum of the edge-weights is below some constant integer, $k$.*

*If we were to restrict the values of the weights in TSP to the integers* 0 *and* 1*, the resulting problem would be equivalent to the Hamiltonian path problem, which is* NP*-hard. This is so because an instance of the Hamiltonian path problem could be reduced to this restricted form of TSP by giving all edges that exist in the initial graph a weight of* 0*, creating all edges that do not already exist in the graph and giving them a weight of* 1*, and setting* $k = 0$*. Therefore, TSP is strongly* NP*-hard.*

On the other hand, if restricting the input values by a polynomial yields a polynomial-time algorithm, the algorithm is said to run in *pseudo-polynomial time*. That is, the algorithms take time $O(p(n, m))$ for all instances where $n$ is the length of the problem instance, $m$ is the largest integer in the instance, and $p$ is some polynomial on $n$ and $m$. These algorithms are said to run in *pseudo*-polynomial time because they are polynomial on the *value* of the input as opposed to the length of the input. One example of such an algorithm is in Section 6.3.

Another way of thinking about pseudopolynomial time algorithms is by changing the encoding of the numbers in the instance to unary (from a

*reasonable* encoding such as binary). Now, if the algorithm runs in polynomial time on the new length (which is really the sum of the values in the instance), then the algorithm runs in pseudo-polynomial time. For a few examples of such algorithms, please refer to Section 7.1. For this reason, one might refer to strongly NP-hard problems as being *unary* NP-*hard* (as mentioned in [16]).

The concepts of strong NP-hardness and pseudo-polynomial time are related to approximation algorithms because [16] proves that in most cases an optimization problem will only have an FPTAS if the problem is not NP-hard in the strong sense. This is because the existence of an FPTAS for a problem implies the existence of a pseudo-polynomial time algorithm for the problem, and the latter algorithm cannot exist if the problem is strongly NP-hard, by definition.

### 4.3.2 *p*-simpleness

However, Garey and Johnson's result is not quite as strong as it could be, as proven by Paz and Moran a year later [35]. Amongst their results is the definition of *p-simple*. A problem in NPO is said to be $p$-simple if it has a pseudo-polynomial algorithm and the maximum value for a given instance has an upper bound which is a polynomial on the length of the input and the optimal solution to the instance. That is, for all instances, $a$, $max(a) = O(p(|a|, opt(a)))$ for some polynomial $p$.

Paz and Moran go on to prove that if there is a fully-polynomial time approximation scheme for a problem, the problem must be $p$-simple. Therefore, in order to prove that a problem cannot be approximated fully polynomially it is sufficient to show that the problem is not $p$-simple. This is stronger than the result in [16] since further qualifications are placed on any problem which has an FPTAS.

### 4.3.3 Other Approximation Results

Ausiello, Crescenzi, and Protasi [3] have written a survey of approximation which contains an introduction to NPO and its subclasses, as well as a fairly up-to-date examination of what is known thus far about approximation.

## 4.4 Creating Approximation Algorithms

Approximating optimal solutions can be an extremely powerful tool in the correct context, especially when the decision form of the optimization prob-

lem at hand is NP-hard in the general case. So how does one go about finding such algorithms, reducing the best-known time complexity dramatically without allowing the rate of error to get out of hand?

### 4.4.1 From Pseudo-Polynomial Time Algorithm to FPTAS

It has been noted that Garey and Johnson showed that the existence of an FPTAS for a problem implies that a pseudo-polynomial algorithm for the problem exists. Therefore, a reasonable first step in the search for a FPTAS for an optimization problem would be to prove first that the problem has a pseudo-polynomial-time algorithm and is thus not strongly NP-hard.

Sometimes it is possible to perform the reverse of this theorem. That is, it tends to be the case that pseudo-polynomial algorithms can be modified to FPTASs.

For one, Woeginger [45] looks specifically at transforming existing pseudo-polynomial dynamic programming algorithms into FPTASs by trying to find a generic structure for dynamic programming algorithms that leads them to be easily transformed to an FPTAS. The two most common approaches mentioned are referred to as *roundingtheinputdata* and *trimmingthestatespace*. The main ideas of these are fairly straightforward. Both use dynamic programming algorithms which are proven optimal but which are only pseudo-polynomial. The former approach then rounds the input data in order to make the algorithm run in true polynomial time, while the latter does not round but rather collapses the number of inputs so that only a polynomial amount of time is used.

# Chapter 5

# Approximating Bribery

Recall from Section 3.4 that bribery is a problem in computational social choice that asks whether, given a budget, it is possible to sway an election toward a preferred candidate and make it a winner of the resulting election. While previous work [15] has shown that this is difficult (i.e., NP-hard) for many voting systems, no work has been done regarding the optimization and approximability of these problems.

As in Chapter 8, where we will study the approximability of some control problems, we will here explore the approximability of bribery problems in their optimization forms. We will present some general nonapproximability results.

## 5.1 Optimizing Bribery

The most natural way to optimize bribery, and the most useful from the point of view of the one doing the bribing, is to minimize the total cost of the bribes. Whereas the bribery decision problems give an election, a preferred candidate, and a budget that cannot be exceeded, when we are optimizing bribery we will only be given the first two and we will find the smallest budget necessary. All of the modifications listed in Section 3.4 (weights varying, prices varying, and both varying) could be optimized.

For any $\mathcal{E}$-bribery problem, let opt-$\mathcal{E}$-bribery refer to the optimization form of the problem. The "opt-" prefix will be added to any modification of the general bribery problem as is the case with the decision problems. For example, $\mathcal{E}$-weighted-bribery becomes opt-$\mathcal{E}$-weighted-bribery, and so on.

As mentioned in Section 4.2.1, we will only be examining approximation for optimization problems with positive optimal solutions. For this reason,

when optimizing bribery, we will only consider instances where each voter's price is a positive integer. Recall that this differs from the standard definition of the decision bribery problem, which allows prices with value 0. In addition to this restriction, we will only consider instances where the preferred candidate is not already winning the instance's election, as otherwise the optimal bribe is 0.

## 5.2 A General Result

As was mentioned in Section 4.3.1, strong NP-hardness of a problem almost always implies that there cannot exist an FPTAS for a problem. This fact immediately brings some answers as to the general approximability of bribery problems.

**Theorem 5.2.1.** *For any voting system $\mathcal{E}$, if $\mathcal{E}$-bribery is* NP*-hard, then opt-$\mathcal{E}$-bribery* $\notin$ FPTAS*, assuming* P $\neq$ NP*.*

*Proof.* This is so because there are no numbers in $\mathcal{E}$-bribery except for the budget. Since if $\mathcal{E}$-bribery is NP-hard the problem will be NP-hard even when the budget is bounded by a polynomial, $\mathcal{E}$-bribery is not a number problem and is by definition strongly NP-hard if it is NP-hard. Since there can be no pseudo-polynomial time algorithms for strongly NP-hard problems and an FPTAS for the problem would imply such an algorithm, there can be no FPTAS for $\mathcal{E}$-bribery □

## 5.3 Manipulation and Bribery

In Section 3.4, we mentioned that Faliszewski, Hemaspaandra, and Hemaspaandra [15] have shown the reducibility of manipulation problems to bribery problems. Here it is worth mentioning the relationship between the two in some detail because the aforementioned reduction is similar to the reductions that will be presented below.

Theorem 4.6 in [15] shows that a manipulation problem within some voting system $\mathcal{E}$ is many-one polynomial-time reducible to the equivalent \$bribery problem in the same system, $\mathcal{E}$. That is, given some instance of the manipulation problem, $M = (C, V, S, p)$, we can efficiently compute an instance, $B$, of the \$bribery problem that will be bribable given the budget if and only if $M$ can be successfully manipulated by the set of voters $S$.

The computed \$bribery instance $B = (C, V_\pi \cup S_\pi, p, 0)$ is made by setting

- $V_\pi$ the same as $V$ with each voter having price $\pi = 1$ and

- $S_\pi$ to be $S$, each voter with price $\pi = 0$ and arbitrary preference lists.

This works since only the voters in $S_\pi$ are allowed to change their preference lists, as the budget is 0. Once the changes are made to the preferences of $S_\pi$, this is equivalent to a manipulation having been performed, since each voter in $S$ in the manipulation instance $M$ would have had to participate in the election as well. Clearly, $M$ and $B$ turn out to be equivalent.

Note that this result also holds when both $M$ and $B$ allow voter weights to vary and when $M$ is unweighted but $B$ is weighted.

## 5.4 Approximation, Unpriced to Priced

Now, using a technique similar to the one in the previous section we can show that some bribery problems will not be approximable. As the reduction in the previous proof essentially made the voters originally in the set $V$ in the manipulation instance unbribable in the bribery instance, our reduction will make it so that any voters in $V$ that are bribed will be detected by a hypothetical approximation algorithm.

This way, any approximation algorithm for the version of the bribery problem must also give an answer to the manipulation decision problem. If the manipulation problem is NP-hard, then a polynomial-time approximation algorithm for the equivalent bribery problem must not exist, assuming $\mathrm{P} \neq \mathrm{NP}$.

At first glance, this would seem like a trivial operation: we could simply use the reduction from the previous section. Then the optimal bribery for our reduced instance will cost 0 if the manipulation instance is manipulable, and, since an the approximate solution must be no greater than $(\epsilon+1)\cdot 0 = 0$, a nonzero approximate solution returned by any approximation algorithm immediately implies that the original instance is not manipulable. This is too simple, of course, since our optimal bribe problems insist upon positive prices, for reasons mentioned earlier.

This being the case, we will still add prices as in the reduction in the previous section, but we will do so in such a way that any solution must be positive.

This brings us to our next theorem.

**Theorem 5.4.1.** *For any voting system $\mathcal{E}$, if $\mathcal{E}$-manipulation is* NP*-hard then opt-$\mathcal{E}$-\$bribery is not approximable (not in* APX*), assuming* $\mathrm{P} \neq \mathrm{NP}$*.*

*Proof.* First, let us reduce an instance $M = (C, V, S, p)$ of $\mathcal{E}$-manipulation to an instance $B = (C, V_\pi \cup S_\pi, p)$ in opt-$\mathcal{E}$-\$bribery. The reduction will

be computed by a function $F(M, \pi_V)$ where $\pi_V \in \mathbb{N}$. As in the previous reduction, we will make the voters in $V_\pi$ too expensive to bribe (in this case, too expensive to bribe without being obvious). That is, the reduction makes it so that

- $V_\pi$ is the same as $V$ with the prices of each voter set to $\pi = \pi_V$ and
- $S_\pi$ is $S$ where each voter is assigned an arbitrary preference list and a price $\pi = 1$.

Now, say we have an approximation algorithm, $A$, for opt-$\mathcal{E}$-\$bribery that runs in polynomial time with constant error $\epsilon \in \mathbb{R}^+$. Then, given an instance $M = (C, V, S, p)$ of the manipulation problem we can calculate $B = F(M, \pi_V) = (C, V_\pi \cup S_\pi, p)$, where $\pi_V = (|S| + 1) \cdot (1 + \lceil \epsilon \rceil)$.

Given $B$, we can calculate its approximate solution, which would be the approximate price of bribing some subset of $V_\pi \cup S_\pi$ such that $p$ is a winner of the resulting election. Let the approximate solution be $z^A = A(B)$. We can now determine whether $M \in \mathcal{E}$-manipulation, via two cases:

- If $z^A < \pi_V$, then the optimal solution, $z^*$, can be no greater than $z^A < \pi_V$. Since the optimal price does not allow even one voter from $V_\pi$ to have been bribed, all of the voters bribed must have come from $S_\pi$. As $S_\pi$ is sufficient to manipulate the election, and it must be likewise for $S$, $M \in \mathcal{E}$-manipulation.
- Otherwise, if $z^A \geq \pi_V$,

$$z^* \geq \frac{z^A}{1+\epsilon} \geq \frac{\pi_V}{1+\epsilon} = \frac{(|S|+1) \cdot (1 + \lceil \epsilon \rceil)}{1+\epsilon} \geq |S| + 1.$$

  Therefore, since the optimal solution spends more than it would cost to bribe all of set $S_\pi$, some voters outside of $S_\pi$ must be used to make $p$ a winner, and similarly $S$ will not be sufficient in $M$ to make $p$ a winner of that election. $M \notin \mathcal{E}$-manipulation.

As we have demonstrated, it would be possible to decide the membership of an instance of an $\mathcal{E}$-manipulation problem in polynomial time given an approximation algorithm for the equivalent opt-$\mathcal{E}$-\$bribery problem. Therefore, if $\mathcal{E}$-manipulation is NP-hard for some voting system $\mathcal{E}$, then opt-$\mathcal{E}$-\$bribery is not approximable. □

As allowing the voters' weights to vary in both the manipulation and bribery problems in the above theorem will not change its proof, this corollary is immediate:

**Corollary 5.4.2.** *For any voting system $\mathcal{E}$, if $\mathcal{E}$-weighted-manipulation is* NP*-hard then opt-$\mathcal{E}$-weighted-$bribery is not approximable (not in* APX*), assuming* P $\neq$ NP*.*

Additionally, since the existence of an approximation algorithm for an $\mathcal{E}$-weighted-$bribery problem would imply the existence of such an algorithm for the equivalent $\mathcal{E}$-$bribery problem, the next corollary is also direct from Theorem 5.4.1.

**Corollary 5.4.3.** *For any voting system $\mathcal{E}$, if $\mathcal{E}$-manipulation is* NP*-hard then opt-$\mathcal{E}$-weighted-$bribery is not approximable (not in* APX*), assuming* P $\neq$ NP*.*

This result, while being rather simple, enables us to make a general statement about the approximability of bribery. This corollary follows directly from Corollary 5.4.2 and Theorem 2.1 from [23] (restated in Section 3.3 as Theorem 3.3.1).

**Corollary 5.4.4.** *For each $m$ and each scoring protocol $\alpha = (\alpha_1, \cdots, \alpha_m)$, if it is not the case that $\alpha_2 = \alpha_3 = \cdots = \alpha_m$, then $\alpha$-weighted-$bribery is not approximable.*

Unfortunately, we do not have evidence that $\alpha$-weighted-$bribery is approximable if $\alpha_2 = \alpha_3 = \cdots = \alpha_m$, which would be the other half of the dichotomy stated in 3.3.1.

# Chapter 6

# The Knapsack Problem

The knapsack problem (KP) is a classic combinatorial problem and one of the first known NP-complete problems. Simply stated, KP is the problem of getting the most benefit while obeying the constraint of a budget. It is this quality of the problem that makes it appealing from the perspective of computational social choice, especially in bribery problems (Section 3.4) where the benefit of bribing a voter (his or her weight in the election) must be weighed against the voter's cost. Specifically, we would like to draw some parallels between KP and plurality-weighed-$bribery, bribing a plurality election where both the weights and prices of voters vary. We will examine this relationship in Section 6.1 in some detail.

Kellerer, Pferschy, and Pisinger [29] and Lin [32] have reasonably extensive and up-to-date listings of the many modifications which have been made to the problem in order to make it applicable in a number of fields and situations, but we will only cover the most common form as well as a few of its modifications here.

This family of problems is applicable in numerous fields, from signal processing [34] to electronic commerce [27]. In addition, the problem's universality and ease of definition are also evident in the appearance of games [41] (however trivial) written around it, as well as the study of forms of the problem modeled after games such as Oregon Trail [9].

## 6.1 Similarity Between Plurality Bribery and the Knapsack Problem

Let us first take a look at the similarities between bribery problems and the knapsack problem, which is suggested by the authors of [15]. Indeed, in

[15], the authors use algorithms similar to those used in KP for this bribery problem. The two problems are surely not equivalent since stealing votes from candidates who are ahead of the preferred candidate may decrease the number of votes that need to be bribed.

For example, one approach toward directly reducing an instance, $B$, of the plurality-weighted-$bribery problem to an instance, $K$, of the Knapsack problem would be by considering the voters (with their vote-weights and prices) in $B$ to be items (with equivalent profits and weights) in $K$. Then, the budget from $B$ becomes the capacity in $K$, and the number of votes by which the preferred candidate is losing becomes the target profit in $K$.

| $K$ | $B$ |
|---|---|
| items | voters |
| profits | vote-weights |
| weights | prices |
| capacity | budget |
| target profit | votes needed in order to win |

But deciding whether the target profit in this knapsack instance is reachable is not the same as deciding the bribability of $B$, as doing so does not take into account the possibility of taking votes from the front-runner(s) in the election and giving them to the preferred candidate. There is no clear, equivalent operation to this in the knapsack problem since taking votes from the winning candidate is essentially reducing the target profit in the knapsack problem.

However, there does seem to be a great deal of similarity between the two, so exploring current approximation algorithms seems worthwhile. In the rest of the chapter, we will look at the definitions of those knapsack problems that seem pertinent along with some of their approximation algorithms, with the goal of bringing some equivalence to the plurality bribery problems to light.

## 6.2 The Binary Knapsack Problem

The knapsack problem asks "given a knapsack of a given size and a number of items with varying sizes and values, is it possible to get a particular amount of profit into the knapsack?" This can be viewed as a generic economic scenario in which a businessman is trying to decide between a number of options, each attributed with costs and profits, and choosing some subset of these options so that he makes enough profit to stay in business.

That is, we are given $n$ items, each with weights ($w_i$) and profits ($p_i$), with $w_i, p_i \in \mathbb{N}$ for all $i = 1, 2, \ldots, n$. We are also given a target profit, $k$, and a capacity, $c$. Now, we decide, is there:

$$x_i \in \{0, 1\} \qquad \text{for} \qquad 1 \leq i \leq n$$

such that

$$\sum_{i=1}^{n} p_i x_i \geq k \text{ and}$$

$$\sum_{i=1}^{n} w_i x_i \leq c$$

Because the values of $x_i$ are restricted to $\{0, 1\}$, this form of the knapsack problem is often referred to as the *Binary* or *0/1* Knapsack Problem. $x_i$ decides whether item $i$ is to be placed in the knapsack or left behind, where a value of 0 leaves the item and a value of 1 takes it. Other forms of the problem allow fractional amounts of an item (*rational* KP) or more than one of an item (*bounded* and *unbounded* KP).

While this notation can make the problem sound awfully complicated, let us look at a simple example of an instance of KP.

**Example 6.2.1.** *Say you are preparing for a short backpacking (knapsacking?) trip, and you have left food out of your knapsack until the end. You will be able to carry* 7 *more pounds, at maximum, but you need to pack food containing at least* 7000 *calories. Here are the items available to you:*

| *Item* | *Pounds* | *Calories* |
|---|---|---|
| *soup* | 3 | 2000 |
| *trail mix* | 1 | 1700 |
| *crackers* | 2 | 1900 |
| *peanut butter and jelly* | 1 | 1500 |
| *bananas* | 3 | 3500 |
| *grapes* | 4 | 4000 |

*Which items should you take to make sure you get enough energy during your hike?*

*Solution.* First, let us look at the problems in terms of the notation of KP.

- the capacity, $c$, is 7,
- the target profit, $k$, is 7000,
- the list of weights will be $w = (3, 1, 2, 1, 3, 4)$, and

- the profits are $p = (2000, 1700, 1900, 1500, 3500, 4000)$.

There are a number of solutions to this instance, but let's say you select soup, trail mix, crackers, and peanut butter and jelly, for a total weight of 7 pounds (which is no greater than our capacity $c$) and 7100 calories (which exceeds our target $k$). In this case, our solution vector (1s when we pack an item, 0s when we leave it) would be set such that exactly the items we picked would be selected: $x = (1, 1, 1, 1, 0, 0)$. □

Some algorithms, both exact and approximate, for this decision formulation of the Binary Knapsack Problem are covered in Section 6.3.

### 6.2.1 KP Optimization

Now, KP is actually a very common problem since profit is rarely encountered without a balancing weight which must be taken into account. But, contrary to the above-stated decision problem, one does not always have a target profit, $k$, in mind, but rather would like to attain as much profit as is possible, given $c$. One might think of a burglar who has gained entry to a home or shop and, only being able to carry a limited weight, must now choose wisely in order to make his trip worthwhile. These types of circumstances are more akin to the optimization form of KP (as opposed to the decision form, above), which does not give a target profit to be attained but rather asks for the highest profit given the capacity of the knapsack.

More formally, we define KP optimization as trying to maximize

$$\sum_{i=1}^{n} p_i x_i$$

where

$$\sum_{i=1}^{n} w_i x_i \leq c$$

$$x_i \in \{0, 1\} \qquad \text{for} \qquad 1 \leq i \leq n$$

$$w_i, p_i \in \mathbb{N} \qquad \text{for} \qquad 1 \leq i \leq n.$$

**Example 6.2.2.** *For instance, if we look at the situation in Example 6.2.1 with the same capacity of 7 pounds, but this time let us find the optimal number of calories (profit) with the given capacity.*

*Solution.* When we look for the maximum number of calories that we can carry, we find that we can get 8600 by taking the trail mix, crackers, peanut butter and jelly, and bananas. As a solution vector, $x$, this would be $x = (0, 1, 1, 1, 1, 0)$. □

This is the more useful form of KP, at least in terms of this thesis, so any future reference to KP (unless otherwise specified) refers to KP optimization.

### 6.2.2 Other Modifications of Binary KP

The plenitude of modifications of the knapsack problem perhaps speaks to its fundamental nature in combinatorics and computer science. While Kellerer, Pferschy, and Pisinger have cataloged many such modifications [29], we will here define only those which are pertinent to the rest of the thesis. We will sometimes refer to the "KP family" of problems, and in doing so intend the problems in this section and the following. All problems in this family have a solution vector ($x$) which is over $\{0, 1\}$ and weights and profits in $\mathbb{N}$.

#### MinKP

The Minimization Knapsack Problem, *MinKP*, is the inverse or the dual of the knapsack problem. Here we have the same givens and attempt to *minimize* the profit such that the cost *exceeds* some value. Or:

$$\text{minimize} \sum_{i=1}^{n} p_i y_i$$

such that

$$\sum_{i=1}^{n} w_i y_i \geq d$$

where $y_i \in \{0, 1\}$ (serving the same purpose as $x_i$ in KP) and $p_i$, $w_i$ are defined as in KP.

To see the relationship between MinKP and KP, it is only necessary to see that solving the equivalent instance of KP—maximizing the profit such that sum of the selected weights is no greater than $c = \sum_{i=1}^{n} w_i - d$—also solves the instance of MinKP. This is why we (and much of the literature) choose to use this counter-intuitive definition where the profit is minimized, for it makes the correlation between KP and MinKP more clear. Given the optimal solution to the KP instance ($x_i$ for $1 \leq i \leq n$), excluding the items selected by this solution and including the rest will give an optimal solution to the MinKP instance ($y_i \neq x_i$ for $1 \leq i \leq n$), and vice versa.

However, an approximately optimal solution to one of the problems does not necessarily yield a solution for the other that is approximate with the same performance ratio, ($\epsilon$). A 1-approximation for MinKP (which doubles as a 1/2-approximation for KP) is presented in [19], which is an application of an FPTAS for the more general problem of Capacitated Plant Allocation [31] (essentially the minimization form of MCKP, below).

**Example 6.2.3.** *Consider again the situation presented in Example 6.2.1, but in this case let us say that you have not yet begun to pack, so there is no capacity. You still want to get* 7000 *calories in your bag, but you just want to do so in the most efficient way.*

*Solution.* Here we would like to minimize the total weight of the items selected. In order to fit this situation with the definition above, we can think of this as either switching the weights and profits in the definition or switching the two in our instance.

Either way, we find that we can pack a little as 6 pounds of food while still getting 7000 calories by packing trail mix, crackers, and bananas. The solution vector for this is $x = (0, 1, 1, 0, 1, 0)$. □

### MCKP

MCKP is the Multiple-Choice Knapsack Problem, wherein the items are partitioned into $n$ groups, or piles, of $m$ items each, and the profit is to be maximized with a given capacity while taking *exactly one* item from each pile.

$$\text{maximize} \sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij} x_{ij}$$

with

$$\sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij} x_{ij} \leq c$$

$$\forall i (1 \leq i \leq n), \sum_{j=1}^{m} x_{ij} = 1.$$

As with KP, $x_{ij} \in \{0, 1\}$ and $p_{ij}$, $w_{ij} \in \mathbb{N}$ for all $i$, $j$, and each retains its meaning from KP. The only difference in the variables is that their subscripts are two-dimensional, $i$ referring to the group and $j$ referring to the item's index in that group.

One might intuitively feel that this is almost KP with the added dimension of groups of items, where the groups are disjoint collections of options dealing with separate decisions which must be solved in the most optimal means. On the other hand, KP is one collection of options for a single decision. This analogy, however, does not entirely hold true since the number of items in a group which can be selected is restricted to one.

**Example 6.2.4.** *Perhaps the most straightforward example of an instance of such a problem would be to imagine that you are packing for a backpacking trip and you want to make sure that you get* exactly one *item from each food group (e.g., fruits and grains), in an attempt to come slightly closer to a sane diet. As before, you still want to maximize the number of calories in total gained by packing the food with some constraint on the weight. Whereas the other problems had only these constraints on weight and calories, an MCKP instance would demand the extra qualification that the food packed has exactly one piece from each kind of food.*

### MDKP

The Multi-Dimensional Knapsack Problem, MDKP, is another generalization of KP. The dimensions referred to in its name are the number of constraints that must hold for a solution to be feasible. For example, if we think of the burglar who is trying to fit the most profitable items in his knapsack, in addition to ensuring that the items fit in the knapsack he might also want to be certain that he is able to lift the total weight of the items. Here, the weights and sizes of the items are two dimensions which are constrained by separate capacities.

In our usual form:

$$\text{maximize} \sum_{i=1}^{n} p_i x_i$$

such that for each dimension, $j$,

$$\sum_{i=1}^{n} w_{ij} x_i \leq c_j$$

and $w_{ij}$, $p_i$, and $x_i$ are defined as usual. It should be clear that when $j = 1$, this is simply KP.

This problem is in general strongly NP-hard [38], see Section 4.3.1 for more about strong NP-hardness.

**Example 6.2.5.** *We could further the instance created in Example 6.2.1 to get an example of an instance of MDKP.*

*Here, you might still be looking for the maximum number of calories, since you need the energy to make it through the hike, and there will still be a limit on the total weight of the food you take. In addition to these we could add constraints on the less-appealing aspects of the food you will be packing, such as fat content and the amount of sodium in them. We would then have to find the amount of fat and sodium in each item, and these would count as weights, each in their own dimension. Then, the problem could be phrased, "how many calories can you pack without the food weighing too much or containing too much total sodium or fat?"*

### KSP

A less well-known variant of KP is KSP, the Knapsack Sharing Problem, concerns, predictably, the "equitable distribution of resources" [8]. The original formulation of KSP by Brown [7] describes the KSP through a real-world example of a manager attempting to evenly distribute raises so that no job class would get left out or given preferential treatment.

Brown's formulation is a little awkward to place in our perception of the KP family thus far, so we choose to adopt what is essentially the notation of Yamada and Futakawa [46] and others, though each is equivalent.

For a given instance of this problem, which consists of groups of items as in MCKP, the goal is to maximize the minimum profit added to a particular group while obeying some weight capacity. Or:

$$\text{maximize} \qquad \min_i \left\{ \sum_{j=1}^{m} p_{ij} x_{ij} \right\}$$

with

$$\sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij} x_{ij} \leq c.$$

Note that, in contrast to MCKP, the number of items taken from a particular group is not restricted to one. In addition, the objective is a maximin function as opposed to a simple maximization. Otherwise, the binary-ness of KP is retained, and $p_{ij}$, $w_{ij}$ are still defined as in MCKP. In fact, if we restrict $n$, the number of groups, to one, then it is easy to see that we are left with the original KP.

**Example 6.2.6.** *Similarly to Example 6.2.4, we might imagine an instance of KSP where we want to get around the same amount of calories from each available food group while obeying the same weight constraint. Here, $w_{ij}$ and $p_{ij}$ represent the weight and profit (calories) of item $j$ in group $i$. Whereas in Example 6.2.4 we were trying to get exactly one item from each group, here we will be looking for a solution that packs items such that the group that has the fewest calories will be larger than it would have been in any other solution. This will often lead to a more even distribution of the calories amongst the groups.*

## 6.3 Algorithms for the Knapsack Problem

Since KP is NP-complete, there are no known true polynomial-time algorithms that are guaranteed to find solutions to instances of KP. While this leaves us with essentially brute-force algorithms, a number of improvements can be had over the most naive brute-force algorithm, and many of these are reviewed in [30]. In addition, there does exist a pseudo-polynomial time dynamic programming algorithm for it. This means that the problem is *not* NP-hard in the strong sense (see Section 4.3.1 for a discussion of this).

### 6.3.1 The Dynamic Programming Algorithm

The traditional dynamic programming algorithm for the binary KP decision problem looks at the optimal solution with a single item and progressively adds an item until all of the items have been considered. For each additional item, the algorithm examines each subcapacity between 0 and $c$. Since the algorithm performs a constant amount of computation at each cell in the table and there are $n$ rows and $c$ columns, the runtime of the algorithm is $O(nc)$.

We will call the dynamic programming matrix $P$ and populate it as follows:

$$P[j,m] = \begin{cases} 0, & \text{if } j = 1 \text{ and } w_j > m \\ p_j, & \text{if } j = 1 \text{ and } w_j \leq m \\ P[j-1,m], & \text{if } j > 1 \text{ and } w_j > m \\ \max\{P[j-1,m], P[j-1,m-w_j]\}, & \text{if } j > 1 \text{ and } w_j \leq m. \end{cases}$$

At each cell, the algorithm checks whether the current item under examination has a profit larger than the current subcapacity. If it does, then the item is irrelevant at this size, so the optimum profit is whatever the

optimum was without the item (i.e., in the row preceding the current one). If the item might fit in the knapsack, it must be allowed a chance to be part of the optimum solution, so the optimum without the item ($P[j-1, m]$) is compared to the optimum if we made room for the item ($P[j-1, m-w_j]$). If we would be better off making room for the item and adding it, then this is the optimal solution.

Once this procedure has run its course, the optimal solution for the problem is the one where all the items are allowed and the subcapacity is equal to $c$—that is, the value in $P[n, c]$.

## 6.4 Approximation Algorithms for KP

### 6.4.1 KP

As the standard maximization form of the knapsack problem is most often studied, there is a great deal of information on approximating it.

#### The greedy approximation algorithm

There is a simple greedy algorithm for KP that approximates the optimal solution by a factor of 1/2, described by Martello and Toth [33]. The algorithm starts by sorting all of the items in the instance in descending order by their efficiencies, that is, their profit-to-weight ratios. Then, the items are added to the sack as they fit until all of the items are tried.

If, at the end, the total profit in the sack is less than the profit of some single item that is not in the sack, that item is placed in the sack alone. This step is added because the simplest greedy algorithm will perform arbitrarily badly otherwise, since there may be a large-profited but inefficient item hiding in the back of the sorted list.

#### Fully-polynomial approximation schemes

One of the first FPTASs (see Section 4.2.3) ever to be defined is described by Ibarra and Kim in [25]. The general idea is to scale or normalize the profits so that the amount of work to be performed is feasible. This algorithm, much as others following it,

1. partitions the items into those with small profits and those with large profits,
2. scales the profits of those items in the large-profited partition down to make them more easily handled,

3. finds the optimal solution to the large (as in profits) partition using the dynamic programming algorithm in Section 6.3.1, and

4. the sack is topped off with the small-profited items using the greedy algorithm.

As the dynamic programming algorithm runs in time pseudo-polynomial (polynomial time if the inputs have a polynomial ceiling), when the items' profits are scaled correctly the algorithm will take polynomial time on the new instance (specifically, in Step 3, above).

This algorithm is of particular interest because it sets the tone for essentially every FPTAS discovered since it was published. A pseudo-polynomial algorithm for a problem is used by first rounding the numbers in the instance that will generally make the algorithm run in super-polynomial time.

Kellerer and Pferschy [28] improve upon Ibarra and Kim's [25] algorithms (as well as a number of interceding improved schemes) time and space complexity greatly. The major difference with this newer scheme is that it breaks the usual partitions into further partitions. Otherwise, the general idea of the original scheme is left intact. This is the FPTAS for KP with the best performance that is known currently.

### 6.4.2 MinKP

There is significantly less information available about approximating the minimization form of KP. In the case of the decision problems or when finding the optimal solutions to KP and MinKP, we can use an algorithm for either KP or MinKP. Here, the solution to the unsolved problem is simply the difference between the sum of the items' profits and the found solution. That is, given an optimal solution, $z^*_{\text{MIN}}$, for an instance of MinKP with target weight $d$, the optimal solution for the equivalent instance of KP (with capacity $c = \sum_{i=1}^{n} w_i - d$ is $z^*_{\text{MAX}} = \sum_{i=1}^{n} p_i - z^*_{\text{MIN}}$.

As such, one might expect an approximate solution to an instance of one problem to similarly yield an approximate solution to the equivalent instance of the other problem. However, this is not necessarily so.

**Example 6.4.1.** *Consider an instance of KP where* $\sum_{i=1}^{n} p_i = 300$ *and the optimal solution is* $z^*_{MAX} = 250$*, so* $z^*_{MIN} = 50$ *for some c, d. If a KP* $1/2$*-approximation algorithm finds an approximate solution* $z^A_{MAX} = 125$ *(the lowest it could be), then we might expect the* $1/2$*-approximate answer to the equivalent MinKP instance to be* $z^A_{MIN} = 300 - z^A_{MAX} = 175$*. Of course, though, this is well over* $150\%$ *of the optimal in the MinKP case, so such an assumption will not be true in general.*

While an approximation for the maximization case will not always serve as an approximation for the minimization case (nor vice versa), there is an algorithm that works well for both. The algorithm, called MinGreedy, is a 1/2-approximation for KP and a 1-approximation for MinKP. It is presented in [19] by Güntzer and Jungnickel.

Since MinGreedy is working from the perspective of MinKP, first the items are sorted in *ascending* order by their efficiencies. Then, the algorithm fills the knapsack with the least efficient items until the next item would make the knapsack reach its target weight (such that the sum of the weights is at least $d$ from our definition in Section 6.2.2). This solution is remembered if the profit is smaller than that arrived at earlier for a feasible solution, otherwise the item that would make the knapsack heavy enough is forgotten about, and the next one is tried. Of course, this loops until all of the items have been processed.

It is relatively surprising that it took until 2000 for such an algorithm to be published, for it is not a difficult algorithm, and it is part of a well-studied problem. We are not aware of an FPTAS for MinKP.

# Chapter 7

# plurality-weighted-$bribery

The problem plurality-weighted-$bribery asks whether it is possible to bribe in a plurality election if the weight (how much a voter's say counts) and price (how much it costs to convince the voter to change his or her preferences) of the voters may vary. This problem is NP-complete according to [15]. While Chapter 5 proves some general results regarding the approximability of bribery, these do not apply to plurality.

While we have not found an approximation algorithm for plurality-weighted-$bribery, we have proved results in Section 7.2 for a similar problem. Then, in Section 7.4 we examine some possible routes toward an approximation algorithm for the problem at hand.

## 7.1 Optimal Algorithms

There are two algorithms in [15] that solve plurality-weighted-$bribery in pseudo-polynomial time (see Section 4.3.1 for a discussion of this concept). This is done by solving the distinct problems plurality-weighted-$bribery$_{\text{unary}}$ and plurality-weighted$_{\text{unary}}$-$bribery, which are special cases of plurality-weighted-$bribery where the prices of the voters and the weights of the voters' votes are respectively encoded in unary rather than some more reasonable encoding such as binary. As it is often the case that pseudo-polynomial algorithms can be used to derive approximation algorithms, let us take a look in some depth at these algorithms, called UnaryPricesBribery and UnaryWeightsBribery.

These algorithms work by using a few dynamic programming algorithms as subprocedures, including one for KP which is also pseudo-polynomial.

The main idea of both algorithms is to establish a threshold, $r$, which $p$,

the preferred candidate, will fall on or above, and the rest of the candidates will be on or under. $r$ is not known in advance for an arbitrary instance of the problem, so it is necessary to iterate over all of the potential thresholds. At each threshold, an attempt is made to bribe enough of the voters who are not voting for $p$ so that $score_E(c) \leq r$ for all other candidates, $c \in C - \{p\}$. If this much is a success, then the algorithms check whether $score_E(p)$ with the addition of the cumulative weight of the bribed voters is greater than or equal to $r$. Should this be the case, it is clearly possible to bribe the voters to make $p$ a winner, so the instance is in plurality-weighted-$bribery.

In the unary prices case, it is possible to iterate over the sum of the voters' prices in the instance since this is a linear operation when they are encoded in unary. Similarly, in the case where we are given unary weights, the sum of said weights can be iterated over in linear time.

#### *heaviest* and *cheapest*

Faliszewski, Hemaspaandra, and Hemaspaandra define the algorithms *heaviest* and *cheapest* in [15] to solve the most basic cases in UnaryPricesBribery and UnaryWeightsBribery, respectively. Given a candidate and (sub)budget, *heaviest* finds the voters (who are currently voting for the candidate) to bribe that will give the most weight without exceeding the budget. On the other hand, *cheapest* finds the least amount of money required to get a cumulative weight of at least some target weight.

If we look at the voters in the election as items in an instance of KP it can be seen that both of these are essentially the dynamic programming algorithm for KP. In *heaviest*, we treat the weights of the voters as the profits of the items in KP and the prices of the voters as the weights of the items in KP. For *cheapest* we do the opposite (treat the prices as profits and vote-weights as weights) and have to convert the instance from one of MinKP (see Section 6.2.2) to standard KP.

In the main algorithms, *heaviest* aids in finding the threshold, $r$, and *cheapest* helps find the value of the remainder of the budget after getting the score of the first candidate down to the threshold.

#### *Heaviest* and *Cheapest*

These two algorithms utilize dynamic programming. *Heaviest* finds the heaviest set of votes that can be bribed from voters voting for some subset of candidates without exceeding a budget and getting all candidates in the subset to have a score not less than or equal to a threshold. *Cheapest* sim-

ilarly finds the cheapest way to bribe votes of weight reaching some target from a subset of candidates while getting their scores down to the threshold.

## 7.2 Plurality Negative Bribery

There is a problem closely related to plurality-weighted-$bribery that is also mentioned in [15]. This problem is pluralityweightednegativebribery, the special case of bribery where a bribe cannot make a voter vote for the preferred candidate, $p$. The weighted version of this is NP-complete, and the problem remains NP-complete even when the number of voters who can be bribed is not restricted, which we can call plurality-weighted-negative-unbounded-bribery (this follows from the reduction from Partition given in [15]).

The difficulty of the unbounded version of the problem leads us to the conclusion that the optimization problem correlating to negative bribery with plurality will not be approximable in the weighted case (and, by corollary, will not be approximable when weights and prices both vary). However, due to our proof technique, this will only hold when the elections are encoded succinctly, as described in Section 3.1.1.

**opt-pluralityweightednegativesuccinctbribery**

**Given:** A set of candidates $C$; a multiset of voters $V$ with full, strict preferences over $C$ and weights $\omega \in \mathbb{N}$ encoded succinctly; and a preferred candidate, $p$.

**Question:** What is the smallest subset of $V$ that can be bribed to vote for any candidate in $C - \{p\}$ and make $p$ a winner of the resulting election?

**Theorem 7.2.1.** *opt-pluralityweightednegativesuccinctbribery* $\notin$ APX*, assuming* P $\neq$ NP*.*

As in the proofs of nonapproximability explored in Chapter 5, we will prove that this is so by starting with an instance of the decision problem and transforming this into an instance of the optimization problem that, if there existed an approximation algorithm for the optimization version with constant error, would give the answer to the original instance.

First, let there be a function, $F$, which takes $\omega_V \in \mathbb{N}$ and $I = (C, V, p)$ in the form of an instance of pluralityweightednegativeunboundedbribery

and returns $I' = (C, V' = \widehat{V} \cup W, p)$ an instance of opt-pluralityweighted-negativesuccinctbribery, where:

- $\widehat{V}$ is identical to $V$ but each voter's weight is multiplied by $\omega_V$
- $W$ is a set of voters of size $|C| \cdot g \cdot \omega_V$, where for each $c_i \in C$, $g \cdot \omega_V$ voters with weight 1 vote for $c_i$, and $g = score_{(C,V)}(a) - score_{(C,V)}(c_i)$ where $a$ is a winner of $(C, V)$.

In the following Lemmas 7.2.3, 7.2.2, and 7.2.4, let us assume that $I = (C, V, p)$ is of the form of pluralityweightednegativeunboundedbribery, and let us assume that $I$ is a *reasonable* instance of this form. Here reasonable means that in election $E = (C, V)$, the combined weight of the votes that have to be taken from those beating $p$ does not exceed the space available of those candidates losing to $p$. To borrow some notation from [15], if $C_{\text{above}} \in C$ is the set of candidates beating $p$ and $C_{\text{below}} \in C$ is the set of candidates losing to $p$, the following must hold for $I$ to be a reasonable instance:

$$\sum_{c \in C_{\text{above}}} (score_E(c) - score_E(p)) \leq \sum_{c \in C_{\text{below}}} (score_E(p) - score_E(c))$$

If an instance is not reasonable in this manner, negative bribery cannot be possible (no matter what the bound on the number of bribes made) since there simply is not enough room for the votes that have to be moved around to make $p$ a winner. Let there also be $I' = F(I, \omega_V) = (C', V' = \widehat{V} \cup W, p)$ for some $\omega_V \in \mathbb{N}$.

**Lemma 7.2.2.** *Given $I$ and $I'$, above, and $V_B$, $\widehat{V}_B$ the results of identical bribes occurring in $V$, $\widehat{V}$. $p$ is a winner in election $E = (C, V_B)$ if and only if $p$ is a winner in election $E' = (C, \widehat{V}_B \cup W)$,*

*Proof.* This is readily clear once one observes that, for any $c \in C$,

$$score_{E'}(c) = score_E(c) \cdot \omega_V + g \cdot \omega_V.$$

Therefore, if $p$ is a winner in $E$ ($score_E(p) \geq score_E(a)$ for any $a \in C - \{p\}$), then $p$'s score in $E'$ will also be at least as large as any other candidate's score in $E'$. Similarly, if $p$ is a winner in $E'$, $p$ must also win $E$. □

**Lemma 7.2.3.** *Given $I$ and $I'$, above, and $E' = (C, V')$, there is always a way to make $p$ win by bribing some subset of $W$.*

*Proof.* One can make $p$ a winner only using the voters in $W$ via the following process. For each candidate $b \in C - \{p\}$ such that $score_{E'}(b) > score_{E'}(p)$, we will bribe $score_{E'}(b) - score_{E'}(p)$ voters from $W$ who are voting for $b$ to vote for the first candidate in $C$ whose score is currently less than $p$'s. If the candidate who gets these votes would have a score greater than $p$'s as a result of these bribes, let the extra votes cascade to the next voter who is losing to $p$, and so on until all of the excess votes have been bribed from $b$.

Since a voter in $V'$ will be beating $p$ by at most $g \cdot \omega_V$ in $E'$, there will always be enough voters in $W$ to bribe using this process to make $p$ a winner. Also, as $I$ is a *reasonable* instance of pluralityweightednegative-bribery, this bribery defined by the above algorithm will always be possible (i.e., there will always be some candidate for the voters in $W$ to lend support to without making that candidate beat $p$). Therefore, some subset of $W$ can always make $p$ a winner of $E'$. □

**Lemma 7.2.4.** *Given $I$ and $I'$, and an optimal negative bribe within $I'$, consisting of voters $\widetilde{V} \subseteq V'$, if $\widetilde{W} = \widetilde{V} \cap W \neq \emptyset$ then $|\widetilde{W}| \geq \omega_V$.*

*Proof.* Assume for contradiction that $|\widetilde{W}|$ is less than $\omega_V$ for some optimal bribe. Let there be elections $E = (C, V)$ and $E' = (C, V')$.

The votes from $\widetilde{W}$ will not be bribed from some $a \in C - \{p\}$ with $score_{E'}(a) > score_{E'}(p)$. Even in the closest situation, where $a$ is the unique winner of $E$ and $E'$ and $score_E(a) - score_E(p) = 1$, then the gap between $a$ and $p$ in $E'$ will be $score_{E'}(a) - score_{E'}(p) = \omega_V$. Even if all of the votes from $\widetilde{W}$ are taken from $a$ and given to some candidate in $C - \{a, p\}$, $a$ will remain the winner.

Similarly, votes from $\widetilde{W}$ will never be bribed from some $b \in C - \{p\}$ with $score_{E'}(b) \leq score_{E'}(p)$. The only reason that this would be attempted would be to bribe these votes from $b$, then bribe a vote to $b$ from some candidate beating $p$ with vote-weight such that it would have made $b$ have a score higher than $p$'s if the votes where not first bribed from $b$.

This will not occur when $|\widetilde{W}| < \omega_V$. Consider the closest case, where $score_{E'}(p) - score_{E'}(b) = (\omega_a - 1) \cdot \omega_V$ (where $\omega_a \cdot \omega_V$ is the weight of the vote that we would like to bribe to $b$). Then, even taking all of the votes from $\widetilde{W}$ and bribing them from $b$ to some candidates in $C - \{b, p\}$, the new difference between the scores of $b$ and $p$ will be

$$\begin{aligned} score_{E'}(p) - score_{E'}(b) + |\widetilde{W}| &= (\omega_a - 1) \cdot \omega_V + |\widetilde{W}| \\ &= \omega_a \cdot \omega_V - \omega_V + |\widetilde{W}| \\ &> \omega_a \cdot \omega_V. \end{aligned}$$

That is, bribing all of the voters in $\widetilde{W}$ with $|\widetilde{W}| < \omega_V$ away from $b$ does not allow votes to be bribed to $b$ that could not have been otherwise, so this will not be done in an optimal bribe.

As bribing fewer than $\omega_V$ voters in $W$ will never make $p$ a winner of the resulting election if it was not already a winner of $E'$, it would be a contradiction to include $\widetilde{W}$ with $|\widetilde{W}| < \omega_V$ in an optimal bribery of instance $I'$. Therefore, an optimal solution to $I'$ will always contain at least $\omega_V$ voters from $W$ if it contains any. □

Now we have enough background to prove our initial theorem, that opt-pluralityweightednegativesuccinctbribery is not approximable.

*Proof of Theorem 7.2.1.* Given an instance $I = (C, V, p)$ in the form of pluralityweightednegativeunboundedbribery and an approximation algorithm $A$ for opt-pluralityweightednegativesuccinctbribery with constant error $\epsilon \in \mathbb{R}^+$, we could decide in polynomial time whether $I \in$ pluralityweighted-negativeunboundedbribery.

First, $I' = F(I, \omega_V) = (C, V' = \widehat{V} \cup W, p)$, where $\omega_V = (|V|+1)\cdot(1+\lceil\epsilon\rceil)$ would be computed. Then we would find the approximate solution to $I'$, $z^A = A(I')$.

If $z^A < \omega_V$, since $z^* \leq z^A$ by Lemma 7.2.4 there cannot be any voters from $W$ in the optimal solution to $I'$, so it must be possible to perform a negative bribery by bribing only voters from $\widehat{V}$, thus by Lemma 7.2.2, it is possible to perform negative bribery on $I$, so $I \in$ pluralityweightednegative-unboundedbribery.

On the other hand, if $z^A \geq \omega_V$, $z^* \geq \frac{z^A}{1+\epsilon} \geq |V| + 1$, so the optimal solution to $I'$ must contain some voters outside of $\widehat{V}$ and by Lemma 7.2.2, there is no subset of $V$ which, when bribed, makes $p$ win $E$. That is, $I \notin$ pluralityweightednegativeunboundedbribery.

As in either case we can decide $I$'s membership regarding plurality-weightednegativeunboundedbribery in polynomial time, such an approximation algorithm *cannot* exist, assuming P $\neq$ NP, since pluralityweighted-negativeunboundedbribery is NP-complete. □

Since an approximation algorithm for opt-pluralityweightednegativesuccinct-$bribery would imply an approximation algorithm for opt-pluralityweighted-negativesuccinctbribery, and the latter is not in APX, the former is not in APX, either.

**Corollary 7.2.5.** *opt-pluralityweightednegativesuccinct$bribery* $\notin$ APX*, assuming* P $\neq$ NP*.*

## 7.3 The Two-Candidate Case

MinGreedy (Section 6.4.2) can be used to approximate opt-pluralityweighted-$bribery when there are only two candidate to consider. In the two-candidate case, we are only concerned with those voters who are voting for the candidate that is not $p$. Therefore, it is a trivial task to take an instance of pluralityweighted$bribery and translate it into an instance of MinKP, where all of the voters not voting for $p$ are turned into items with weight equal to their price and profit equal to their vote-weight, and the target profit is half the difference between the score of the other candidate and the score of $p$.

Then, when MinGreedy is given this instance as input (after exchanging profits and weights so that weight, and thus voter price, is minimized) it will return a price no more than twice the price that it would cost to make $p$ win through bribery.

## 7.4 Attempted approximations

Despite the fact that bribery in plurality with both weights and prices varying is approximable in the two-candidate case, we know that many other bribery problems are not approximable, and we are still not sure whether there exists an approximation algorithm for pluralityweighted$bribery with three or more candidates. Now, we will look at some attempts to derive approximation in the general case.

### 7.4.1 Modifying Existing, Optimal Algorithms

One particularly straightforward attempt toward creating an approximation scheme for opt-pluralityweighted$bribery would include modifying one of the pseudopolynomial time algorithms given in [15] and covered in Section 7.1. These are *UnaryPricesBribery* and *UnaryWeightsBribery*, in which the prices or weights, respectively, are given as input unary encoded. These algorithms show that pluralityweighted$bribery is not strongly NP-hard, for the algorithms are in P for the problem when the *values* of the prices or weights are used (as opposed to the encoded lengths of the quantities) to measure the complexity of the problem.

We might attempt to use UnaryPricesBribery [15]. It uses a subprocedure which borrows heavily from dynamic programming solutions to KP. Even if this subprocedure, *Heaviest* were replaced by one of the FPTASs for KP, with the appropriate reduction occurring beforehand, our resulting algorithm would not be an FPTAS for pluralityweighted$bribery. This is

so because there is a loop in UnaryPricesBribery which runs $k$ times, where $k$ is the budget for the given instance. $k$ is $O(2^n)$ for instances encoded in any reasonable encoding, so this simple formulation will not be usable as an FPTAS. But in UnaryPricesBribery, if $k \geq \pi(V)$ (where $\pi(V)$ is the sum of the prices in the instance), the algorithm immediately accepts. Since $k < \pi(V) < n$ will hold when the prices are *unary encoded*, this loop does not keep the algorithm from running in pseudopolynomial time.

We might also look at UnaryWeightsBribery [15]. This uses subroutines *Cheapest* and *cheapest* which are dynamic programming solutions similar to those used in the unary prices case. However, there is one rather significant difference because *cheapest* is really an instance of MinKP, not KP. See Section 7.1 for more on MinKP and Section 6.2.2 to see that an approximation algorithm for MinKP is not necessarily one for KP. There are known approximation algorithms for MinKP, but this fact is inconsequential since UnaryWeightsBribery loops over the sum of the weights of the voters, and this would lead any approximation algorithm based simply on replacing *Cheapest*/*cheapest* with an approximation algorithm for MinKP would still not necessarily be in P.

Therefore, the two most straightforward approaches toward an approximation algorithm for opt-pluralityweighted$bribery appear not to be feasible (that is, unless some means of trimming the search space in the loops was found), and so it is necessary to explore other means. The above approaches are most desirable since rounding one of the inputs (essentially restricting the input values to a polynomial) is a simple and often-used means of creating an FPTAS for a problem.

### 7.4.2 Greedy Heuristics

As the above attempts to translate pseudo-polynomial algorithms into FPTASs for this problem failed, we looked at some greedy heuristic approaches. This tends to be a common way of coming to an algorithm that puts an upper bound on the optimal solution for a problem (for instance, greedy algorithms for KP).

The naive greedy heuristics essentially look at the voters in an election as a set of items in an instance of KP. The simplest of these, which we will call *NG1*, is the greedy heuristic for KP which sorts all items in descending order by their efficiency (in our case, $\omega(v_i)/\pi(v_i)$) and bribes each voter in this order until $p$ becomes a winner.

Next, let us call another heuristic *NG2*, which is basically the same as *NG1*, but when sorting by the efficiencies, a voter's $\omega$ is multiplied by two

if the voter is currently picking a winner of the election. This change also means that the order could change drastically between individual bribes, so the sorting is done after each bribe. This could easily be changed to only do so after a winning candidate is caused to no longer be winning by a bribe.

### Other Greedy Heuristics

Some other potential greedy heuristics work on the basis of bribing sets of voters, each set making $score_E(p)$ at least as high as some candidate who was previously beating $p$. Here, the idea is to continually make progress in $p$'s ranking, where in the naive heuristics above many bribes could be performed without achieving any progress, in this sense. In addition, the above-presented *NG2* would be rather shortsighted when $p$ is losing to more than one candidate. In *NG2*, the likelihood of selecting a voter is only increased when that voter is currently picking an overall winner, but this tactic could obviously lead to bad choices, and in some cases it makes sense to give some sort of weight to votes going toward any candidate currently beating $p$.

We present a family of greedy heuristics. They consist of three general steps:

Step 1 Use the KP greedy algorithm to find the cost of bribing voters such that $p$ will be beating $c$ for each $c \in C$ such that $score_E(c) > score_E(p)$.

Step 2 Bribe those votes associated with the minimum value from Step 1.

Step 3 Repeat Steps 1 and 2 until $p$ becomes a winner.

So, in the simplest case, this heuristic only considers the votes that are currently going to $c$ in Step 1. The $\omega$s of these votes are counted double since the vote will be stealing weight from $c$ and adding weight to $p$.

In a variation, in addition to considering the votes currently going to $c$ in Step 1, all votes going to candidates not beating $p$ are also considered (but their $\omega$s are only considered singly, while those of the votes currently going to $c$ are still considered doubly). The rationale here is that, while those extra votes are in general a distraction from the goal of beating $c$, there may be some cheap ones that are worthwhile.

Another variation could change Step 1 from the one in the previous modification to include all votes going to some candidate other than $c$ beating $p$ in addition to the rest. These votes are counted singly as if they were votes

for a candidate losing to $p$. Again, the goal at each iteration is to make $p$ overcome some candidate, $c$.

One characteristic of this family that makes it feel more effective than the more naive greedy methods is that the former bribe sets of voters at a time rather than one vote at a time. The latter take one vote at a time and continually change the weight that votes get as a result. These circumstances make members of this family feel more stable to some degree and, perhaps, more predictable for an arbitrary instance (whether this is the case or not).

At any rate, none of these heuristics appear to put an upper bound on the error between the found solution and the optimal solution.

# Chapter 8

# Approximating Control

As mentioned in Section 3.2, the problem of control in voting systems involves looking at the election from the point of view of one who can modify the election to his or her liking, and from this point of view attempting to make a particular candidate win or lose.

For some voting systems and methods of control, it has been proven by either [5] or [21] that the underlying language for doing so is NP-hard (or, in the terminology of [5, 21], the system is *resistant* to a particular method of control). While, assuming P $\neq$ NP, this is a reasonable assurance that the control cannot generally be performed on an instance of an election using that particular voting scheme, one might wonder how difficult it is to find or approximate optimal solutions to such problems. As usual, the reasoning goes that, while coming to the exact solution for a problem may be NP-complete, finding an approximate solution within some error bound and in a reasonably small amount of time can be both a desirable outcome and a more feasible one.

We now present some results regarding some of the problems traditionally studied, along with some modifications on these.

## 8.1 Adding Candidates

Traditionally, control by adding candidates is a decision problem where, given an election and a set $D$ of spoiler candidates, one attempts to make a distinguished candidate, $c$, win or lose the election by adding candidates from $D$. Of course, for this to work, the preferences of the voters in the election must cover $D$ as well as the candidates initially in the election. In the deletion case, one tries to delete candidates from the set of candidates

$C$ in order to make $c$ win or lose.

Let us call the problem of destructive control by adding candidates in the plurality voting system *plurality-$AC_D$* for short, and for the constructive case we will use *plurality-$AC_C$*.

It can also be interesting to ask how much of $D$ must we add to make $c$ lose? Or, how much of $C$ must we remove to make $c$ lose? These are the optimization problems related to the initial decision problem (which we will call *opt-plurality-$AC_D$* and *plurality-$AC_C$*, respectively), and they give us something we might be able to approximate.

Let us present some preliminary results regarding the approximability of these problems:

**Lemma 8.1.1.** *Neither* opt-plurality-$\text{AC}_D$ *nor* plurality-$\text{AC}_C$ *is a number problem. Both are strongly* NP*-hard.*

*Proof.* Neither problem allows voter weights, so an instance of either will simply be sets of candidates and voters, neither of which needs to be represented using numbers—certainly not arbitrarily large numbers. The decision forms of both are NP-complete [21], and any problem which is NP-hard but not a number problem is strongly NP-hard, by definition [16]. □

**Theorem 8.1.2.** *Neither* opt-plurality-$\text{AC}_D$ *nor* plurality-$\text{AC}_C$ *is in* FPTAS*, assuming* $\text{P} \neq \text{NP}$*.*

*Proof.* As the existence of an FPTAS for a problem implies the existence of a pseudo-polynomial algorithm for the same problem, and there can be no pseudo-polynomial algorithm for a problem that is *strongly* NP-hard, if $\text{P} \neq \text{NP}$, there can be no FPTAS for destructive control by adding or deleting candidates in plurality. This is proven in [16]. □

### 8.1.1 Weighted Cases

As previously mentioned, it can be useful to look at the cases in elections where each voter has a weight assigned to it. Here, these cases are useful because the problems are number problems, and thus are not immediately strongly NP-hard as in Lemma 8.1.1. We will learn much about the unweighted cases by looking first at these weighted cases.

#### Weighted and Destructive

Let us define *opt-plurality-$AC_{DW}$*, the optimization problem for destructively adding candidates to a plurality election where the voters have weights, as follows:

**Given:** A set $C = \{c, c_1, \ldots, c_{m-1}\}$ of qualified candidates containing a distinguished candidate $c$, a set $D = \{d_1, \ldots, d_\ell\}$ of possible spoiler candidates, and a set $V = \{v_1, \ldots, v_k\}$ of voters with preferences, *prefs* over $C \cup D$ and weights $\omega \in \mathbb{N}$.

**Question:** What is the smallest $\widetilde{D} \subseteq D$ that can be added to the election $(C, V)$ and assure that $c$ is no longer a unique winner of the resulting election, $(C \cup \widetilde{D}, V)$?

Now let us propose a statement with conclusions stronger than is posed in Theorem 8.1.2:

**Theorem 8.1.3.** *Optimal destructive control by adding candidates to a plurality election where the voters are weighted (*opt-plurality-AC$_{DW}$*) is not approximable (not in* APX*), assuming* P $\neq$ NP*.*

We are going to prove Theorem 8.1.3 by contradiction, but, first, let us explain the idea behind the proof and prove some intermediate results.

**Proof Idea** We will write a function that takes an instance $I$ in the form of *plurality-AC$_D$* and outputs an instance $I' \in$ *opt-plurality-AC$_{DW}$*, where a number of extra candidates and voters have been added and weights have been added to all of the preexisting voters. Once we have this new instance, we will show that any approximation algorithm for *opt-plurality-AC$_{DW}$* necessarily decides *plurality-AC$_D$*. As the approximation algorithm and our function are both in polynomial time, their combination is a polynomial time algorithm for *plurality-AC$_D$*, which cannot exist assuming P $\neq$ NP, since *plurality-AC$_D$* is NP-hard.

In order to make $I'$ optimizable, we must make it so that there are *always* some spoiler candidates who can make the distinguished candidate not the unique winner (as, if there is no set that will be successful, there is nothing to optimize). This is assured by Lemma 8.1.6.

In order to accomplish this, we must add some new candidates and voters that will always be able to make the distinguished candidate lose, but, at the same time, we want the original set of spoiler candidates to be able to change the outcome of the new election in a way equivalent to the way they do in the old one (Lemmas 8.1.4 and 8.1.5). We will add the new candidates and voters in such a way that if the new spoiler candidates are added to the election in the approximate solution, so many would need to be added (with respect to the approximation algorithm's error bound) that the approximation algorithm betrays the fact that these new candidates must

be used (Lemma 8.1.7). If this occurs, the original set of spoiler candidates is inadequate, so we know that the original instance is not in *plurality-AC*$_D$. If none of the new candidates are used, the approximate solution will be smaller with respect to its error bound, so we will know that the original spoilers are sufficient—the original instance is in *plurality-AC*$_D$.

Now, we present some intermediate results.

Let $F_W$ be a function from an instance $I = (C, D, V, c)$ (in the proper form of an instance of *plurality-AC*$_D$) and $\omega_V \in \mathbb{N}$ to $I' = (C', D', V', c) \in$ *opt-plurality-AC*$_{DW}$, defined as follows:

- $C' = C \cup \widehat{D}$,
- $D' = D \cup F$,
- $V' = \widehat{V} \cup V_1 \cup V_2 \cup V_3$, and
- $c$ remains as it was in $I$,

where the extra candidates $\widehat{D}$ and $F$ are defined as

- $\widehat{D} = \{\hat{d}_1, \ldots, \hat{d}_{|D|}\}$ and
- $F = \{f_1, \ldots, f_{g \cdot \omega_V}\}$ where $g = score_{(C,V)}(c) - score_{(C,V)}(a)$ where $a$ is the candidate with the next highest score in the election $(C, V)$

and the new voters $\widehat{V}$, $V_1$, $V_2$, and $V_3$ are defined as

- $\widehat{V} = \{\hat{v}_1, \ldots, \hat{v}_{|V|}\}$, where $\hat{v}_i = v_i$ with weight $\omega = \omega_V$ and all candidates in $\widehat{D} \cup F$ added to the end of $v_i$'s *prefs*,
- $V_1 = \{v_1, \ldots, v_{|F|}\}$, each with $\omega = 1$ and for each $1 \leq i \leq |F|$, $v_i$'s *prefs* $= f_i > c > \ldots$ over $C' \cup D'$ where $f_i \in F$,
- $V_2 = \{v_1, \ldots, v_{|C|-1}\}$, each with $\omega = g \cdot \omega_V$ and for each $1 \leq i \leq |C|-1$, $v_i$'s *prefs* $= c_i > \ldots$ over $C' \cup D'$ where $c_i \in C - \{c\}$, and
- $V_3 = \{v_1, \ldots, v_{|D|}\}$, each with weight $\omega = g \cdot \omega_V$ and for each $1 \leq i \leq |D|$, $v_i$'s *prefs* $= d_i > \hat{d}_i > \ldots$ over $C' \cup D'$ where $d_i \in D$ and $\hat{d}_i \in \widehat{D}$.

For the following Lemmas 8.1.4, 8.1.5, 8.1.6, and 8.1.7, let us assume there exists $I = (C, D, V, c)$ and $I' = F_W(I, \omega_V) = (C', D' = (D \cup F), V' = (\widehat{V} \cup V_1 \cup V_2 \cup V_3), c)$ for some $\omega_V \in \mathbb{Z}^+$.

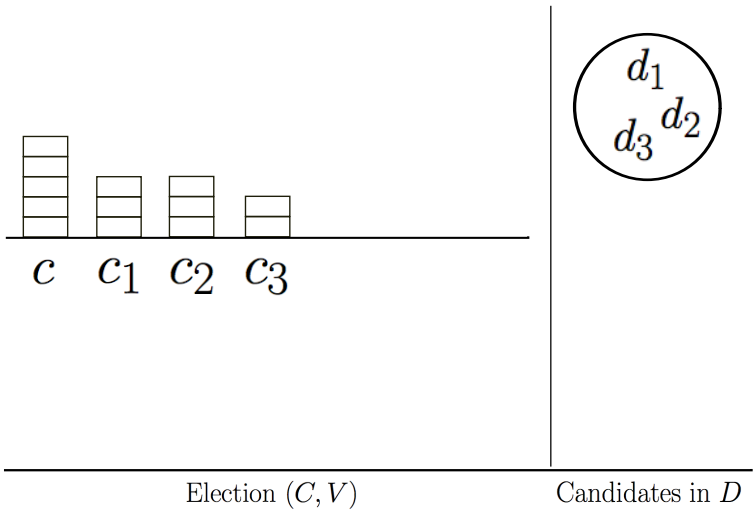


Figure 8.1: An initial instance of *plurality-AC*$_D$.

**Lemma 8.1.4.** *If $\widetilde{D} \subseteq D$, the scores of any candidate $a \in C \cup \widetilde{D}$ in elections $E = (C \cup \widetilde{D}, V)$ and $E' = (C' \cup \widetilde{D}, V')$ are related as follows:*

$$score_{E'}(a) = g \cdot \omega_V + score_E(a) \cdot \omega_V.$$

*Proof.* As for each $v_i \in V$ that votes for $a$ in $E$, the equivalent $\hat{v}_i \in \widehat{V}$ will vote for $a$ in $E'$, and since each $\hat{v}_i$ has weight $\omega_V$, $score_{(C' \cup \widetilde{D}, \widehat{V})}(a) = score_E(a) \cdot \omega_V$.

Now to account for the $g \cdot \omega_V$ term. There are exactly three places $a$ could fall: $a = c$, $a \in C - \{c\}$, or $a \in \widetilde{D}$.

In the first case, $a = c$, all of the voters in $V_1$ will vote for $a$ in $E'$, giving $a$ weight $g \cdot \omega_V$. This is so because no candidates from $F$ are present in $E'$, so the first preference for each member of $V_1$ will be $c$.

In the next case, $a \in C - \{c\}$, a voter in $V_2$ with weight $g \cdot \omega_V$ will vote for $a$ in $E'$, by definition of $I'$.

Finally, when $a \in \widetilde{D}$, one of the voters in $V_3$ will give $a$ $g \cdot \omega_V$ in $E'$.

As each case leaves $a$ with $g \cdot \omega_V + score_E(a) \cdot \omega_V$ weight in election $E'$ and there are no other voters to be considered, this will always be the case for $a \in C \cup \widetilde{D}$. □

**Lemma 8.1.5.** *A set of candidates $\widetilde{D} \subseteq D$ exists such that $c$ is not the unique winner of election $E = (C \cup \widetilde{D}, V)$ if and only if $c$ is not the unique*

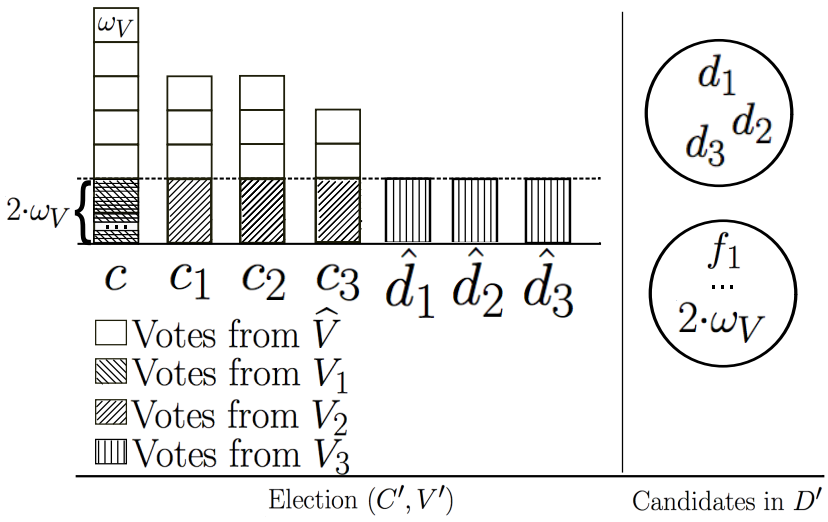


Figure 8.2: The output of $F_W$ when given as input the instance from Figure 8.1 and some $\omega_V$.

*winner in election* $E' = (C' \cup \widetilde{D}, V')$.

*Proof.* ($\Longrightarrow$) If $c$ is not the unique winner of election $E$, then $c$ will not be the unique winner of $E'$. Since $c$ is not the unique winner of $E$, there must be some $b \in (C - \{c\}) \cup \widetilde{D}$ such that

$$score_E(b) \geq score_E(c).$$

Since by Lemma 8.1.4, $score_{E'}(b) = g{\cdot}\omega_V + score_E(b){\cdot}\omega_V$ and $score_{E'}(c) = g \cdot \omega_V + score_E(c) \cdot \omega_V$, clearly $score_{E'}(b) \geq score_{E'}(c)$, so $c$ is not a unique winner of $E'$.

($\Longleftarrow$) If $c$ is not the unique winner of election $E'$, then $c$ will not be the unique winner of $E$. As in the converse, when $c$ is not the winner of $E'$, we know that there is some candidate $b \in (C - \{c\}) \cup \widetilde{D}$ such that

$$score_{E'}(b) \geq score_{E'}(c),$$

so by Lemma 8.1.4

$$g \cdot \omega_V + score_E(b) \cdot \omega_V \geq g \cdot \omega_V + score_E(c) \cdot \omega_V,$$

which clearly implies that $score_E(b) \geq score_E(c)$, that is, $c$ is not the unique winner of $E$.

As both implications hold, some $\widetilde{D} \subseteq D$ exists such that $c$ is not the unique winner of $E$ if and only if the same $\widetilde{D}$ ensures that $c$ is not the unique winner in $E'$. In other words, there is a subset of $D$ that, when added to the election implied by the original instance, makes $c$ lose if and only if $c$ loses when the same subset is added to the election of the transformed instance. □

**Lemma 8.1.6.** *Candidate $c$ will not be the unique winner in election $(C' \cup F, V')$.*

*Proof.* Recall, by the definition of $g$ in $F_W$,

$$score_{(C,V)}(c) - score_{(C,V)}(a) = g$$

for some $a \in C - \{c\}$, and, by Lemma 8.1.4,

$$\begin{aligned} score_{(C',V')}(c) &= g \cdot \omega_V + score_{(C,V)}(c) \cdot \omega_V \\ score_{(C',V')}(a) &= g \cdot \omega_V + score_{(C,V)}(a) \cdot \omega_V \\ &= \omega_V \cdot (g + score_{(C,V)}(a)) \\ &= score_{(C,V)}(c) \cdot \omega_V \end{aligned}$$

thus

$$score_{(C',V')}(a) = score_{(C',V')}(c) - g \cdot \omega_V.$$

Now, when we consider the election $(C' \cup F, V')$, all votes $v_i \in V_1$ which contribute to $score_{(C',V')}(c)$ will instead vote for $f_i \in F$, as this is how each $v_i$'s *prefs* is defined. Since $|F| = g \cdot \omega_V$ and each, when added to the election, will take 1 unit of weight from $c$,

$$\begin{aligned} score_{(C' \cup F, V')}(c) &= score_{(C',V')}(c) - g \cdot \omega_V \\ &= score_{(C',V')}(a) \\ &= score_{(C' \cup F, V')}(a). \end{aligned}$$

Therefore, $c$ is not a unique winner in $(C' \cup F, V')$. □

**Lemma 8.1.7.** *In addition to $I$ and $I'$, let $\widetilde{D}$ be the smallest subset of $D'$ such that $c$ is not the unique winner of $(C' \cup \widetilde{D}, V')$. If $\widetilde{F} = \widetilde{D} \cap F \neq \emptyset$, $|\widetilde{F}| \geq \omega_V$.*

*Proof.* Assume for contradiction that we have an optimal $\widetilde{D}$ and $\widetilde{F}$ as described above, and $|\widetilde{F}| < \omega_V$.

Recall that by the definition of $V'$, the only voter that could give vote-weight to a candidate $f_i \in F$ on addition of $f_i$ to an election with voters $V'$ is a member of $V_1$, and that each of these voters originally (in election $(C', V')$ gives vote-weight 1 to $c$. Each member of $F$ has only the power to remove vote-weight 1 from $c$; it neither has the ability to win an election nor to remove weight from any other candidate in $C' \cup D' - \{c\}$.

For any $a \in (C' - \{c\}) \cup \widetilde{D}$ there exists some $t \in \mathbb{Z}$ such that

$$score_{(C' \cup \widetilde{D} - \widetilde{F}, V')}(c) = score_{(C' \cup \widetilde{D} - \widetilde{F}, V')}(a) + t \cdot \omega_V.$$

But if $t \geq 1$ ($c$ beats $a$ in election $(C' \cup \widetilde{D} - \widetilde{F}, V')$) and $|\widetilde{F}| < \omega_V$ (by our assumption),

$$score_{(C' \cup \widetilde{D}, V')}(c) > score_{(C' \cup \widetilde{D}, V')}(a),$$

that is, $c$ will beat $a$ regardless of the addition of $\widetilde{F}$, and this is a contradiction since $\widetilde{D}$ cannot not be optimal.

Therefore, if $c$ wins $(C' \cup \widetilde{D} - \widetilde{F}, V')$, $c$ also wins election $(C' \cup \widetilde{D}, V')$. Since $\widetilde{F}$ is unnecessary, $\widetilde{D}$ is trivially smaller by the exclusion of $\widetilde{F}$, so $\widetilde{F}$ would never be part of a smallest such $\widetilde{D}$. □

Now we are able to prove our main theorem, that *opt-plurality-AC*$_{DW}$ is not approximable.

*Proof of Theorem 8.1.3.* We will prove Theorem 8.1.3 by contradiction by assuming that there is an approximation algorithm for *opt-plurality-AC*$_{DW}$, $A$. On input $I' = (C', D', V', c)$, $A$ returns an approximate solution $z^A$ such that $z^* \leq z^A \leq (1+\epsilon)z^*$, where $z^*$ is the optimal solution to $I'$ and $\epsilon \in \mathbb{R}^+$ is a constant. Recall that these optimal and approximate solutions are in terms of the size of the smallest (and approximately smallest, respectively) subset $\widetilde{D} \subseteq D'$ that will make $c$ not the unique winner of election $(C' \cup \widetilde{D}, V')$.

But if there was such an $A$, given an instance $I = (C, D, V, c)$ in the form of the problem *plurality-AC*$_D$, $A$ could be used to find an optimal solution to $I$. First, $I' = F_W(I, \omega_V = (1 + \lceil \epsilon \rceil) \cdot |D| + 1) = (C', D' = D \cup F, V', c)$ would be generated, and the approximate solution $z^A = A(I')$ would be computed.

By Lemma 8.1.6, we know that even when there is no subset $\widetilde{D}$ of $D$ that makes $c$ lose election $(C \cup \widetilde{D}, V)$ that there will always be some $D'' \subseteq D'$ that will make $c$ lose $(C' \cup D'', V')$, so $z^*$ and $z^A$ will always be positive integers, and we know that we only need worry about the following two cases:

- $z^A \leq (1 + \lceil \epsilon \rceil) \cdot |D|$, then we know

$$z^* \leq (1 + \lceil \epsilon \rceil) \cdot |D| < \omega_V.$$

  By Lemma 8.1.7, this implies that no subset of $F$ is present in the optimal solution to $I'$, that is, some $\widetilde{D} \subseteq D$ exists such that $c$ is not the unique winner in election $(C' \cup \widetilde{D}, V')$. Then, by Lemma 8.1.5, we know that this $\widetilde{D}$ exists if and only if $c$ is also not the unique winner of $(C \cup \widetilde{D}, V)$. Therefore, $I \in$ *plurality-AC*$_D$.

- $z^A > (1 + \lceil \epsilon \rceil) \cdot |D|$ on input $I'$. Then

$$z^* \geq \frac{z^A}{1+\epsilon} > \frac{(1+\epsilon) \cdot |D|}{1+\epsilon} > |D|,$$

  so some subset of $F$ must be included in an optimal solution to $I'$. Thus we know that not even adding all of $D$ to the election can make $c$ lose—otherwise, that would be the optimal solution. That is, $c$ will win election $(C' \cup D, V')$ and election $(C \cup D, V)$ by Lemma 8.1.5, and, therefore, $I \notin$ *plurality-AC*$_D$.

In sum, if there was such an $A$ for *opt-plurality-AC*$_{DW}$, by taking an instance $I$ of the form of a problem in *plurality-AC*$_D$, we could determine in polynomial time whether or not $I \in$ *plurality-AC*$_D$ by finding $I' = F_W(I, \omega_V)$ (since $\epsilon$ is a constant, $F_W$ will be polynomial) and using $I'$ as input to $A$. Assuming P $\neq$ NP, this is a contradiction, since *plurality-AC*$_D$ is NP-hard. Therefore *opt-plurality-AC*$_{DW}$ is not approximable, *opt-plurality-AC*$_{DW} \notin$ APX. □

Following directly from Theorem 8.1.3:

**Corollary 8.1.8.** opt-plurality-AC$_D$ *is not approximable (not in* APX*), assuming* P $\neq$ NP.

*Proof.* The same proof will hold if we make $F_W$ add a number of voters equal to the weight of the voters added (rather than adding a single voter with its respective weight in the weighted case) to the new instance. □

### Weighted and Constructive

As above, we will define *plurality-AC*$_{CW}$, the optimization problem for constructively adding candidates to a plurality election where the voters have weights, as

**Given:** A set $C = \{c, c_1, \ldots, c_{m-1}\}$ of qualified candidates containing a distinguished candidate $c$, a set $D = \{d_1, \ldots, d_l\}$ of possible spoiler candidates, and a set $V = \{v_1, \ldots, v_k\}$ of voters with preferences over $C \cup D$ and weights $\omega \in \mathbb{N}$.

**Question:** What is the smallest $\widetilde{D} \subseteq D$ that can be added to the election and assure that $c$ is a unique winner of the resulting election, $(C \cup \widetilde{D}, V)$?

We will prove the same result as in the destructive case, that is:

**Theorem 8.1.9.** *Optimal constructive control by adding candidates to a plurality election where the voters are weighted (*plurality-AC$_{CW}$*) is not approximable (not in* APX*), assuming* P $\neq$ NP*.*

**Proof Idea** As in the previous proof, we will define a construction that will take an unweighted instance and return a weighted instance that will betray the answer to the former with a polynomial-time approximation algorithm.

As such, the method is rather similar to that used for *opt-plurality-AC$_{DW}$* (Theorem 8.1.3), with the crucial difference that the extra candidates and voters put into the weighted instance will be created so that they will be able to remove weight from any candidate who is not $c$, our distinguished candidate (as opposed to being able to remove weight from $c$ in the destructive case).

Now let us present some intermediate results.

Let $F_W$ be a function from an instance $I = (C, D, V, c)$ (in the proper form of an instance of *plurality-AC$_C$*) and $\omega_V \in \mathbb{N}$, to $I' = (C, D' = D \cup \widehat{D}, V' = \widehat{V} \cup W \cup \{v_c\}, c) \in$ *plurality-AC$_{CW}$*, where:

- $\widehat{D} = (\hat{d}_{11}, \ldots, \hat{d}_{ij}, \ldots, \hat{d}_{|C-\{c\}|\ell})$ where $\ell = (g \cdot \omega_V + \omega_V - 1)$ and $g = score_{(C,V)}(a) - score_{(C,V)}(c) + 1$ such that $a \in C - \{c\}$ is a winner of election $(C, V)$,
- $\widehat{V} = \{\hat{v}_1, \ldots, \hat{v}_{|V|}\}$, where for each $1 \leq i \leq |V|$, $\hat{v}_i = v_i$ with weight $\omega = \omega_V$ and all candidates in $\widehat{D}$ added to the end of $v_i$'s *prefs*,
- $W = \{w_{ij} \mid c_i \in C - \{c\}$ and $1 \leq j \leq g \cdot \omega_V + \omega_V - 1$ and $\omega = 1$ and $prefs = \hat{d}_{ij} > c_i > \ldots$ and $\hat{d}_{ij} \in \widehat{D}\}$, and
- $v_c$ has weight $g \cdot \omega_V$ and $prefs = c > \ldots$.

For the following Lemmas 8.1.10, 8.1.11, 8.1.12, and 8.1.13, let there be defined $I = (C, D, V, c)$ and $I' = F_W(I, \omega_V) = (C, D' = D \cup \widehat{D}, V' = \widehat{V} \cup W \cup \{v_c\}, c)$ for any $\omega_V \in \mathbb{N}$.

**Lemma 8.1.10.** *Let there be elections $E = (C \cup \widetilde{D}, V)$ and $E' = (C \cup \widetilde{D}, V')$ for any $\widetilde{D} \subseteq D$. If $b \in C - \{c\}$, $score_{E'}(b) = score_E(b) \cdot \omega_V + g \cdot \omega_V + (\omega_V - 1)$. Also, $score_{E'}(c) = score_E(c) \cdot \omega_V + g \cdot \omega_V$.*

*Proof.* This is so because if $b \in C - \{c\}$, by definition of $F_W$, each point gained in $E$ by $b$ is matched by a vote-weight of $\omega_V$ in $E'$ by the corresponding voter in $\widehat{V}$, thus $score_{(C,\widehat{V})}(b) = score_E(b) \cdot \omega_V$. In addition, the voters $W$ will lend $g \cdot \omega_V + \omega_V - 1$ points to such a $b$, for a final score of $score_{E'}(b) = score_E(b) \cdot \omega_V + g \cdot \omega_V + (\omega_V - 1)$.

For $c$, the initial $score_E(c) \cdot \omega_V$ points also come from the set $\widehat{V}$. The remaining points are due to the voter $v_c$. □

**Lemma 8.1.11.** *Let there be elections $E = (C \cup \widetilde{D}, V)$ and $E' = (C \cup \widetilde{D}, V')$ for any $\widetilde{D} \subseteq D$. $c$ is the unique winner of $E$ if and only if $c$ is the unique winner of $E'$.*

*Proof.* ($\Longrightarrow$) If $score_E(c) > score_E(a)$, then $score_{E'}(c) > score_{E'}(a)$. By Lemma 8.1.10,

$$score_{E'}(c) = (score_E(c) + g) \cdot \omega_V$$

and

$$score_{E'}(a) = (score_E(a) + g) \cdot \omega_V + \omega_V - 1.$$

Since $score_E(c) \geq score_E(a) + 1$,

$$\begin{aligned} score_{E'}(c) &\geq (score_E(a) + 1 + g) \cdot \omega_V = (score_E(a) + g) \cdot \omega_V + \omega_V \\ &> score_{E'}(a). \end{aligned}$$

($\Longleftarrow$) If $score_{E'}(c) > score_{E'}(a)$, then $score_E(c) > score_E(a)$. This is so since by Lemma 8.1.10,

$$score_E(c) \cdot \omega_V + g \cdot \omega_V > score_E(a) \cdot \omega_V + g \cdot \omega_V + \omega_V - 1$$

which implies

$$score_E(c) \cdot \omega_V > score_E(a) \cdot \omega_V + \omega_V - 1$$

so certainly $score_E(c) > score_E(a)$. □

**Lemma 8.1.12.** *Let there be elections $E = (C, V)$ and $E' = (C, V')$. There exists some $\widetilde{D} \subseteq \widehat{D}$ such that $c$ wins $(C \cup \widetilde{D}, V')$.*

*Proof.* As $c$ could not lose any voters by adding $\widetilde{D}$ to the election,

$$score_{(C \cup \widetilde{D}, V')}(c) = score_{E'}(c) = score_E(c) \cdot \omega_V + g \cdot \omega_V + (\omega_V - 1)$$

by Lemma 8.1.10. If $a \in C$ is a winner of $E$,

$$\begin{aligned} score_{E'}(c) &= score_E(c) \cdot \omega_V + g \cdot \omega_V + (\omega_V - 1) \\ &= score_E(c) \cdot \omega_V + (1 + score_E(a) - score_E(c)) \cdot \omega_V + (\omega_V - 1) \\ &= (1 + score_E(a)) \cdot \omega_V + \omega_V - 1. \end{aligned}$$

Let us take $\widetilde{D} = \widehat{D}$, that is, let us add all candidates in $\widehat{D}$ to election $E'$. Since $score_{(C \cup \widehat{D}, V')}(a) = score_{E'}(a) - (g \cdot \omega_V + \omega_V - 1)$, by Lemma 8.1.10 $score_{(C \cup \widehat{D}, V')}(a) = score_E(a)$, which is clearly less than $score_{(C \cup \widehat{D}, V')}(c)$. Therefore, $c$ will uniquely win $(C \cup \widehat{D}, V')$. □

**Lemma 8.1.13.** *Let there also be elections $E = (C \cup \widetilde{D}, V')$ and $E' = (C \cup \widetilde{D} \cup G, V')$, where $\widetilde{D} \subseteq D$ and $G \subseteq \widehat{D}$. If $\widetilde{D} \cup G$ is an optimal solution to $I'$ and $G \neq \emptyset$, $|G| \geq \omega_V$.*

*Proof.* Say $0 < |G| < \omega_V$. Then $G$ will, at best, take $\omega_V - 1$ points from a single candidate in $C - \{c\}$. Say these points affect entirely $a \in C - \{c\}$, where $a$ is a winner of $E$.

Then $score_{E'}(a) = score_E(a) - (\omega_V - 1)$. Since $a$ is a winner of $E$, Lemma 8.1.10 implies that

$$\begin{aligned} score_E(a) &\geq score_E(c) + (\omega_V - 1) \\ score_{E'}(a) &\geq score_{E'}(c), \end{aligned}$$

so $c$ cannot possibly win $E'$ unless it also uniquely wins $E$ and therefore the inclusion of $G$ has no bearing on the outcome of $E'$ with respect to $c$ and will not be included in an optimal solution to $I'$. □

Now we are able to prove Theorem 8.1.9.

*Proof of Theorem 8.1.9.* Assume there is an approximation algorithm, $A$, which finds approximate solutions to instances of *plurality-AC*$_{CW}$ in polynomial time with error bounded by some constant $\epsilon \in \mathbb{R}^+$. Given an instance $I = (C, D, V, c)$, we can in polynomial time derive $I' = F_W(I, \omega_V = (|D| + 1) \cdot (1 + \lceil \epsilon \rceil) = (C, D' = D \cup \widehat{D}, V' = \widehat{V} \cup W \cup \{v_c\}, c)$. $F_W$ will

only take polynomial time since $\epsilon$ is a constant—$\omega_V$ is simply written as a parameter to make the earlier proofs simpler. We can then find the approximate solution to $I'$, $z^A = A(I')$, the approximate size of the subset of $D'$ that needs to be added to the election in order to make $c$ a unique winner.

If $z^A < \omega_V$, $I \in$ *plurality-AC*$_C$. Since the optimal solution, $z^*$ is at most $z^A$, the corresponding solution could not contain any subset of $\widehat{D}$, by Lemma 8.1.13. Therefore, it must be possible to make $c$ the unique winner by adding only candidates from $D$.

Otherwise if $z^A \geq \omega_V$, $I \notin$ *plurality-AC*$_C$. As $z^* \geq \frac{z^A}{1+\epsilon} \geq |D| + 1$, $z^* > |D|$, that is, there *must* be some candidates outside of $D$ used in the optimal solution to $I'$, so $I \notin$ *plurality-AC*$_C$ since there is no subset of $D$ that will make $c$ a winner, by Lemma 8.1.11.

Since $F_W$ and $A$ can determine whether $I$ is in *plurality-AC*$_C$ or not in polynomial time and *plurality-AC*$_C$ is NP-hard, $A$ cannot exist, that is, *plurality-AC*$_{CW} \notin$ APX. □

Since we can use unweighted voters in the previous set of proofs equivalently by adding one identical voter for each unit of weight in the construction (as in Corollary 8.1.8), the following is immediate:

**Corollary 8.1.14.** plurality-AC$_C$*, the optimization problem for constructively controlling a plurality election by adding unweighted voters, is not approximable (not in* APX*), assuming* P $\neq$ NP*.*

## 8.2 Deleting Candidates

We can define the respective optimization problems for both destructive and constructive control by deleting candidates under the plurality voting scheme. When examining the unweighted cases of these problems, they are clearly not number problems (for the same reason as in Lemma 8.1.1) and thus not in FPTAS (as in Lemma 8.1.2).

However, it appears that we are not able to use the same technique as in the proofs of Theorems 8.1.3 and 8.1.9. This is so because the related decision problems (*plurality-AC*$_D$ and *plurality-AC*$_C$) have no bounds besides the spoiler candidates used to perform the control must be a subset of a given set. In a sense, the candidate-adding problems are *unbounded*, whereas problems regarding candidate deletion are *bounded*: there is some $k$ and the number of candidates deleted must not exceed it.

The immediate problem awaiting one who attempts to write a proof of nonapproximability of the optimization problems for deleting candidates is

that, while it is reasonably easy to tell if some candidates outside of those in the original instance (i.e., those added when constructing the new instance) are used in the approximate solution, it is difficult to tell whether more than $k$ of the candidates in the original instance were used to find the approximate solution.

When the decision problems are unbounded (such as in control by adding candidates), we simply scale the weights of the preexisting voters to give significantly more weight to the preexisting candidates than to our additional candidates, and we can do this because we do not care how many of the preexisting candidates have to be added. However, in bounded cases such as candidate deletion, since each preexisting candidate is scaled the same way using similar techniques as before, it is not possible (or, at least, not as simple) to determine how many of such candidates were used in the approximate solution.

## 8.3 Partitioning

As with the other problems, the partitioning control problems, including partitioning candidates and voters and the variations within these, have associated optimization problems. These are less obvious than the others, but they are natural. We will define an optimum partition to be one in which the size of the largest half resulting from the partition is minimized.

As with all of the other control problems, these are not number problems and are not in FPTAS as long as their respective decision problems are NP-hard. For plurality, this means that any partition of candidates optimization will not be in FPTAS and partition of voters with ties promoting is also not in FPTAS.

As we can see in Table 3.1, control by partition is often NP-hard in the plurality system and is sometimes the same in approval and Condorcet when partitioning the voters. Proving that control by partition is not approximable is difficult, and the techniques from the previous proofs are of little use here. The biggest difficulty encountered while trying to apply such techniques to these problems is that it is much more difficult to modify the election in a controlled way with partitioning. Also, controlling how the partitions happen (and ensuring that *bad* partitions are noticed) is problematic.

| | Plurality | | Condorcet | | Approval | |
|---|---|---|---|---|---|---|
| Control by | Construct. | Destruct. | Construct. | Destruct. | Construct. | Destruct. |
| Adding Candidates | SR | SR | I | V | I | V |
| Deleting Candidates | WR | WR | V | I | V | I |
| Partition of Candidates | TE: WR<br>TP: WR | TE: WR<br>TP: WR | V | I | TE: V<br>TP: I | TE: I<br>TP: I |
| Run-off Partition of Candidates | TE: WR<br>TP: WR | TE: WR<br>TP: WR | V | I | TE: V<br>TP: I | TE: I<br>TP: I |
| Adding Voters | V | V | WR | V | WR | V |
| Deleting Voters | V | V | WR | V | WR | V |
| Partition of Voters | TE: V<br>TP: WR | TE: V<br>TP: WR | WR | V | TE: WR<br>TP: WR | TE: V<br>TP: V |

Table 8.1: Summary of results regarding the approximability of control problems for plurality, Condorcet, and approval voting schemes, with *unweighted* voters as presented in this chapter. I = immune, WR = weakly resistant (not in FPTAS), SR = strongly resistant (not in APX), V = vulnerable, TE = Ties-Eliminate, TP = Ties-Promote. Vulnerability and immunity results follow directly from [5, 21].

## 8.4 Adding and Deleting Voters

For plurality, neither of these problems are difficult, but are rather both in P.

However, these are difficult for approval and Condorcet voting in the constructive cases, and if their unbounded versions were also NP-hard it would be possible to show that these are not approximable, by using techniques such as is used in the proofs for control by adding candidate in plurality (Section 8.1). Since both addition and deletion are easy in approval and Condorcet in the unbounded case (where as many candidates as liked may be added or deleted), such a proof will not work, for reasons similar to those given in Section 8.2.

Since none of these problems are number problems, none of them will be in FPTAS, whether or not voter weights are allowed to vary.

## 8.5 Summary

To get a better idea of the texture of the approximability of control problems, let us construct tables similar to Table 3.1 (which enumerates the complexities of each constructive and destructive case for control problems) in Section 3.2.2. The results are Table 8.1 and 8.2, which contain the tightest known bounds regarding the approximability of the control problems, in their unweighted and weighted versions, respectively.

We have included one table each for the unweighted and weighted versions of these problems because it is unclear whether they will be identical.

| | Plurality | | Condorcet | | Approval | |
|---|---|---|---|---|---|---|
| Control by | Construct. | Destruct. | Construct. | Destruct. | Construct. | Destruct. |
| Adding Candidates | SR | SR | I | V | I | V |
| Deleting Candidates | WR | WR | V | I | V | I |
| Partition of Candidates | TE: WR<br>TP: WR | TE: WR<br>TP: WR | V | I | TE: V<br>TP: I | TE: I<br>TP: I |
| Run-off Partition of Candidates | TE: WR<br>TP: WR | TE: WR<br>TP: WR | V | I | TE: V<br>TP: I | TE: I<br>TP: I |
| Adding Voters | V | V | WR | V | WR | V |
| Deleting Voters | V | V | WR | V | WR | V |
| Partition of Voters | TE: V<br>TP: WR | TE: V<br>TP: WR | WR | V | TE: WR<br>TP: WR | TE: V<br>TP: V |

Table 8.2: Summary of results regarding the approximability of control problems for plurality, Condorcet, and approval voting schemes, with *weighted* voters as presented in this chapter. I = immune, WR = weakly resistant (not in FPTAS), SR = strongly resistant (not in APX), V = vulnerable, TE = Ties-Eliminate, TP = Ties-Promote. Vulnerability and immunity results follow directly from [5, 21].

If the unweighted version of a problem is strongly resistant, then the more general weighted version must also be strongly resistant. If there exists an approximation algorithm for a weighted control problem, it must also perform an approximation when all of the voters' weights are unit weights, that is, in the unweighted case. The same argument can be made regarding weak resistance.

However, we cannot necessarily always transplant approximability resistance results from a weighted control problem to the unweighted problem. We have seen in Corollaries 8.1.8 and 8.1.14 that sometimes nonapproximability of an unweighted problem can be inferred from the nonapproximability of the equivalent weighted problem.

In the cited instances, the relationship is easy to see because if there existed an approximation algorithm for the unweighted problem, an instance of the weighted problem could be approximated by making it unweighted (adding, say $\omega$ identical voters for a voter with $\omega$ weight, though we must be careful to avoid exponential weights—in these cases, the weights in question are dependent only on $\epsilon$, not the length of the instance) and using the unweighted approximation algorithm.

But when we look at performing such a reduction for voter addition, deletion, or partition, the $\omega$ voters in the unweighted instance corresponding to one weighted voter might not be controlled in the same manner, thus vastly changing the mechanics of the problem. It is possible, and perhaps likely, that there exist reductions for these voter-control problems from the weighted to the unweighted cases, say, by cleverly adding candidates in such a way that the full set of unweighted voters must be all controlled

the same way. These reductions look to be as involved as those proving nonapproximability.

To be clear: it may be the case that some unweighted control problems (specifically, voter-control problems) are approximable (but not in PO) even if their corresponding weighted problems are not approximable.

# Bibliography


[1] Aikaterini, K. Analysis and Comparison of the Greek Parliamentary Electoral systems of the period 1974-1999. Master's thesis, Athens University of Economics and Business, 2000.

[2] Arrow, K. J. *Social Choice and Justice*, vol. 1 of *Collected Papers of Kenneth J. Aarow*. Belknap Press, Cambridge, MA, 1984.

[3] Ausiello, G., Crescenzi, P., and Protasi, M. Approximate solution of NP optimization problems. *Theoretical Computer Science 150*, 1 (1995), 1–55.

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

[5] Bartholdi, J. J. I., Tovey, C. A., and Trick, M. A. How hard is it to control an election? *Mathematical and Computer Modeling*, 16 (1992), 27–40.

[6] Bovet, D., and Crescenzi, P. Introduction to the theory of complexity. Available at `http://pamela.dsi.unifi.it/piluc/mod/resource/view.php?id=39`, Jun. 2006.

[7] Brown, J. R. The Knapsack Sharing Problem. *Operations Research 27*, 2 (Mar.–Apr. 1979), 341–355.

[8] Brown, J. R. The Sharing Problem. *Operations Research 27*, 2 (Mar.–Apr. 1979), 324–340.

[9] Burg, J., Ainsworth, J., Casto, B., and Lang, S. Experiments with the 'Oregon Trail Knapsack Problem'. `http://www.cs.wfu.edu/~burg/papers/knapsack.doc`. Accessed 15 Jan. 2007.

[10] Chamberlin, J. R., and Cohen, M. D. Toward Applicable Social Choice Theory: A Comparison of Social Choice Functions under Spatial Model Assumptions. *The American Political Science Review 72*, 4 (Dec. 1978), 1341–1356.

[11] Christian, R., Fellows, M., Rosamond, F., and Slinko, A. On Complexity of Lobbying in Multiple Referenda. to be published in *Review of Economic Design*, 2006.

[12] Conitzer, V. Computing Slater Rankings Using Similarities Among Candidates. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06)* (2006).

[13] Conitzer, V., Lang, J., and Sandholm, T. When Are Elections with Few Candidates Hard to Manipulate? to be published in the *Journal of the ACM*.

[14] Conitzer, V., and Sandholm, T. Universal Voting Protocol Tweaks to Make Manipulation Hard. In *Proceedings of the International Joint Conference on Artificial Intelligence* (8 2003), pp. 781–788.

[15] Faliszewski, P., Hemaspaandra, E., and Hemaspaandra, L. A. The Complexity of Bribery in Elections. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI 2006)* (Jul. 2006), AAAI Press, pp. 641–646.

[16] Garey, M. R., and Johnson, D. S. "Strong" NP-completeness results: Motivation, examples, and implications. *J. ACM 25*, 3 (1978), 499–508.

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

[18] Gibbard, A. Manipulation of voting schemes: A general result. *Econometrica* (1973).

[19] Güntzer, M. M., and Jungnickel, D. Approximate minimization algorithms for the 0/1 Knapsack and Subset-Sum Problem. *Operations Research Letters 26* (2000), 55–66.

[20] Hägele, G., and Pukelsheim, F. The electoral writings of Ramon Llull. *Studia Lulliana 41*, 97 (2001), 3–38.

[21] Hemaspaandra, E., Hemaspaandra, L., and Rothe, J. Anyone but Him: The Complexity of Precluding an Alternative. to be published in *Artificial Intelligence*.

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

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

[24] Homan, C. M., and Hemaspaandra, L. A. Guarantees for the Success Frequency of an Algorithm for Finding Dodgson-Election Winners. In *Proceedings of the 31st International Symposium on Mathematical Foundations of Computer Science* (Aug. 2006).

[25] Ibarra, O. H., and Kim, C. E. Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems. *J. ACM 22*, 4 (1975), 463–468.

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

[27] Kameshwaran, S. *Algorithms for Piecewise Linear Knapsack Problems with Applications in Electronic Commerce*. PhD thesis, Indian Institute of Science, 2004.

[28] Kellerer, H., and Pferschy, U. A New Fully Polynomial Time Approximation Scheme for the Knapsack Problem. *Journal of Combinatorial Optimization*, 3 (1999).

[29] Kellerer, H., Pferschy, U., and Pisinger, D. *Knapsack Problems*. Springer-Verlag, Berlin, 2004.

[30] Kreher, D. L., and Stinson, D. R. *Combinatorial Algorithms: Generation, Enumeration, and Search*. CRC Press, London, 1999.

[31] Labbé, M., Schmeichel, E. F., and Hakimi, S. L. Approximation algorithms for the capacitated plant allocation problem. *Operations Research Letters 15* (1994), 115–126.

[32] Lin, E. Y.-H. A bibliographical survey on some well-known non-standard knapsack problems. *INFOR 36*, 4 (1998), 274–317.

[33] Martello, S., and Toth, P. *Knapsack Problems*. Discrete Mathematics and Optimization. John Wiley & Sons, New York, 1990.

[34] Mohr, A. E. Bit Allocation in Sub-linear Time and the Multiple-Choice Knapsack Problem. In *DCC '02: Proceedings of the Data Compression Conference (DCC '02)* (Washington, DC, USA, 2002), IEEE Computer Society, p. 352.

[35] Paz, A., and Moran, S. Non Deterministic Polynomial Optimization Problems and Their Approximations. *Theoretical Computer Science 15*, 3 (1981), 251–277.

[36] Pisinger, D. *Algorithms for Knapsack Problems*. PhD thesis, University of Copenhagen, 1995.

[37] Procaccia, A. D., Rosenschein, J. S., and Zohar, A. Multi-winner elections: Complexity of manipulation, control and winner-determination. In *The Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007)* (Hyderabad, India, Jan 2007), pp. 1476–1481.

[38] Puchinger, J., Raidl, G. R., and Pferschy, U. The Core Concept for the Multidimensional Knapsack Problem. In *Evolutionary Computation in Combinatorial Optimization* (Budapest, Hungary, 2006), Lecture Notes in Computer Science, pp. 195–208.

[39] Rabinowitz, G., and Macdonald, S. E. A Directional Theory of Issue Voting. *The American Political Science Review 83*, 1 (Mar. 1989), 93–121.

[40] Reny, P. J. Arrow's Theorem and the Gibbard-Satterthwaite Theorem: A Unified Approach. *Economics Letters 70*, 1 (2001), 99–105.

[41] Richardson, L. The Knapsack Problem: The Game of Premature Optimization. `http://www.crummy.com/software/if/knapsack/`, Apr. 2004. Accessed 15 Jan. 2007.

[42] Saari, D. G. The symmetry and complexity of elections. `http://www.colorado.edu/education/DMP/voting_b.html`. Accessed 4 Apr. 2007.

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

[44] Vazirani, V. V. *Approximation Algorithms*. Springer, Berlin, 2003.

[45] Woeginger, G. J. When does a dynamic programming formulation guarantee the existence of an FPTAS? In *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 1999), Society for Industrial and Applied Mathematics, pp. 820–829.

[46] Yamada, T., and Futakawa, M. Heuristic and reduction algorithms for the knapsack sharing problem. *Computers and Operations Research 24*, 10 (1997), 961–967.