# Bribery over Social Networks

BTP Project-II report submitted to

Indian Institute of Technology Kharagpur

in partial fulfilment for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

**by**

**Abhinav Sen**

**(20CS10003)**

**Under the supervision of**

**Professor Palash Dey**


**Department of Computer Science and Engineering**

**Indian Institute of Technology Kharagpur**

**Spring Semester, 2023-24**

**May 02, 2024**

# DECLARATION

I certify that

(a) The work contained in this report has been done by me under the guidance of my supervisor.

(b) The work has not been submitted to any other Institute for any degree or diploma.

(c) I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

(d) Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Date: May 02, 2023 (Abhinav Sen)

Place: Kharagpur (20CS10003)

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR**

**KHARAGPUR - 721302, INDIA**


## *CERTIFICATE*

This is to certify that the project report entitled "**Bribery over Social Networks**" submitted by **Abhinav Sen** (Roll No. 20CS10003) to Indian Institute of Technology Kharagpur towards partial fulfilment of requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering is a record of bona fide work carried out by him under my supervision and guidance during Spring Semester, 2023-24.

Date: May 02, 2023

Place: Kharagpur

Professor Palash Dey
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur - 721302, India

# *Abstract*

---

Name of the student: **Abhinav Sen** Roll No: **20CS10003**

Degree for which submitted: **Bachelor of Technology**

Department: **Department of Computer Science and Engineering**

Thesis title: **Bribery over Social Networks**

Thesis supervisor: **Professor Palash Dey**

Month and year of thesis submission: **May 02, 2024**

---

Bribery has long been recognized as a pervasive issue in the context of elections and decision-making processes. This thesis delves into the realm of bribery over social networks, with a specific focus on understanding and optimizing resource allocation in an environment where voters are interconnected, represented as a graph. Each voter in this network possesses an associated cost function, defining the expense required to bribe them. We then look for algorithms to find optimal bribery schemes under a predefined budget to successfully change the course of the election. We do this for special cases and constraints as well to study tractable solutions.

This research contributes to the understanding of resource allocation under bribery schemes in interconnected social networks. By addressing the complexity and algorithmic aspects of the problem, we offer insights into mitigating the influence of bribery in decision-making processes while considering the intricacies of social network structures. The findings of this thesis can potentially inform policy decisions, election strategies, and algorithmic solutions to combat bribery within the modern world.

# *Acknowledgements*

We would like to express our sincere gratitude to the following individuals, institutions, and organizations whose contributions and support were instrumental in the completion of this research paper.

First and foremost, we extend our deepest appreciation to our research supervisor, Palash Dey, whose guidance, expertise, and unwavering support have been invaluable throughout the entire research process.

Our heartfelt thanks go to the staff and facilities at IIT Kharagpur for providing the necessary resources, library access, and research assistance. Your institution's support played a pivotal role in shaping this research.

Abhinav Sen

# Contents

# Chapter 1

# Motivation

## 1.1 Introduction

Election Bribery and control are very important topics, not just for human agents, but for computer agents as well as seen in (Chakraborty et al., 2019), (Ephrati and Rosenschein, 1991), (Pennock et al., 2000) etc While previous works exist on control and bribery in voting, none of them consider the inter-relationships of the actors being manipulated. This makes them highly detached from real life, where inter-relationships are highly important in manipulation strategies.

## 1.2 Literature Survey

The motivation for studying complexity issues comes from a result known as the Gibbard–Satterthwaite Theorem, with additional results that have been established since (most notably the Duggan–Schwartz Theorem), showing that every reasonable election system can be manipulated (see (Gibbard, 1973), (Satterthwaite, 1975), (Duggan and Schwartz, 2000), (Taylor, 2005)]) given complete information. One of the insights that naturally drove Bartholdi, Tovey, and Trick (Bartholdi et al., 1989) to study complexity issues is that even if an election system has wonderful mathematical properties, if determining who won under the election system is computationally intractable then that system isn't going to be practically useful. So

better design of election systems cannot prevent manipulation. Bartholdi, Tovey, and Trick (Bartholdi et al., 1989) proposed getting around this obstacle by seeking to make manipulation exorbitantly expensive, computationally

A major collection of results in contemporary computational social choice can be found in F. Brandt and Procaccia (2015). A lot of important its results can be derived from here as well (Bartholdi et al., 1989) and (Bartholdi III et al., 1992), which were the precursors to the study of social choice computationally. The papers talk about the complexity of various bribery schemes under various voting rules. For the purpose of this thesis, we mostly look at the majority and plurality voting rule, which always select the condorset winner (F. Brandt and Procaccia, 2015), if it exists. From (Bartholdi III et al., 1992) we can see that WEIGHTED SHIFT BRIBERY problem is $\mathcal{NP}$-complete for the plurality rule, while the SHIFT BRIBERY with plurality is polynomial solvable. There are a lot of different settings under which one can study complexity for any voting rule. For example- homogeneous-vs.-nonhomogeneous electorate bribability, bounded-size-vs.-arbitrary-sized candidate sets, weighted-vs.-unweighted voters, and succinct-vs.-nonsuccinct input specification, etc (Faliszewski et al., 2009). From (Bredereck et al., 2016) we can see that Combinatorial Shift Bribery problem is $\mathcal{NP}$-complete and W[1]-hard even for the Plurality rule for only five voters and two candidates and no budget constraints. For the broad class of elections (including plurality, Borda, k-approval, and veto) known as scoring protocols, (Faliszewski et al., 2009) proves a dichotomy result for bribery of weighted voters: finding a simple-to-evaluate condition that classifies every case as either NP-complete or in P. (Elkind and Faliszewski, 2010) discusses a 2 approximation algorithm for a large class of problems under the copeland and maxmin rules.

But the problem also has further layers of complexity. There are cases where systems immune or computationally resistant to a chair choosing the winner nonetheless are vulnerable to the chair blocking a victory ((Hemaspaandra et al., 2007)). Interestingly, the paper cites the fact that destructive control is often much easier than constructive control, although often with less incentive. Further, there are also other forms of control. For example, setting political boundaries (gerrymandering) can cause election manipulation as seen in (Ito et al., 2019). Moving agents across political boundaries or voting spaces can also have the same effect ((Lev and Lewenberg, 2019)). The idea of modelling geographical distances between voters graphically can

be seen in (Cohen-Zemach et al., 2018)- gerrymandering over graphs. The problem comprises of dividing a network into components such that the preferred candidate wins in a plurality of these components. This problem too was proved to NP hard in the general case

Also, the complexity classes of several problems in voting is heavily dependant on the bounded nature of the number of candidates. ((Conitzer and Sandholm, 2002)) shoed that for many problems under different scoring rules, there exists a small number of fixed candidates for which the bribery problem becomes NP-hard. This is further discussed in (Conitzer et al., 2003), which shows interesting results on exactly how many candidates are needed for different types of bribery problems to be computationally challenging.

Clearly, the class or problems that arise from social choice theory are often non trivial in nature. There has been a lot of work on the computational complexity of manipulation. The resistance of voting rules to constructive or destructive manipulation is often measured in terms of the computational difficulty of manipulation. However, there isn't a lot of work considering the inter-relationships of voters being bribed. We attempt a more generalised version of the problems studied so far.

## 1.3 Research gaps

While the computational complexity of various bribery schemes under various voting rules have been studied, none of these studies have taken into account the interactions between the voters being bribed. For example, it might be sensible to bribe a voter with high influence, who can also impact other voters. There have been models that attempt to aggregate voters (for example, using graphical ties to represent geographical distances between voters (Bentert et al., 2021)). However, that has so far mostly been done in the context of political districting for manipulation, which is more of control manipulation than bribery per se. We, on the other hand, model the influence of one voter on others through a weighted graph of influences. This produces a more holistic and general model to the bribery problem that is more closely tied to real life.

## 1.4 Objective

The primary objective of this project is to explore the complexity and develop algorithms to identify optimal resource allocation strategies under a bribery scheme within interconnected social networks. By leveraging the relationships between voters as represented by the graph structure, we aim to answer critical questions pertaining to the efficiency, fairness, and ethical implications of such resource allocation schemes.

The thesis provides a rigorous analysis of the computational complexity of this problem. We investigate the fundamental questions of whether the problem is NP-hard, and if so, we aim to determine the boundaries of its complexity. We also explore special cases and constraints that can lead to tractable solutions.

# Chapter 2

# Preliminaries

## 2.1 Introduction

Let us look at some commonly used terminology in the study of control and bribery

## 2.2 Commonly used terminology

An election is a pair $(\mathcal{C}, \mathcal{V})$, where $\mathcal{C} = \{c_1, \ldots, c_m\}$ is a set of candidates and $\mathcal{V} = \{v_1, \ldots, v_n\}$ is a set of voters. If not mentioned otherwise, we use $n$ and $m$ to respectively denote the number of voters and candidates. Each voter $v_i$ has a preference order (vote) $\succ_i$, which is a linear order over $\mathcal{C}$. We denote the set of all complete orders over $\mathcal{C}$ by $\mathcal{L}(\mathcal{C})$. We call a list of $n$ preference orders $\{\succ_1, \succ_2, \ldots, \succ_n\} \in \mathcal{L}(\mathcal{C})^n$ an $n$-voter preference profile. We denote the $i^{\text{th}}$ preference order of a preference profile $\mathcal{P}$ as $\succ_i^{\mathcal{P}}$. A map $r : \mathcal{L}(\mathcal{C})^n \to \mathcal{C}$ is called a resolute voting rule (as we assume the unique-winner model); in case of ties, the winner is decided by a *lexicographic* tie-breaking mechanism $\succ_t$, which is some pre-fixed ordering over the candidates. For a set of candidates $X$, let $\overrightarrow{X}$ be an ordering over them. Then, $\overleftarrow{X}$ denotes the reversed ordering of $\overrightarrow{X}$.

Let $[\ell]$ represent the set of positive natural numbers up to $\ell$ for any positive integer $\ell$. A voting rule $r$ is called *anonymous* if, for every preference profile $(\succ_i)_{i\in[n]} \in \mathcal{L}(\mathcal{C})^n$

and permutation $\sigma$ of $[n]$, we have $r((\succ_i)_{i\in[n]}) = r((\succ_{\sigma(i)})_{i\in[n]})$. A voting rule is called *efficient* if the winner can be computed in polynomial time of the input length.

A scoring rule is induced by an $m$-dimensional vector, $(\alpha_1, \ldots, \alpha_m) \in \mathbb{Z}^m$ with $\alpha_1 \geqslant \alpha_2 \geqslant \ldots \geqslant \alpha_m$ and $\alpha_1 > \alpha_m$. A candidate gets a score of $\alpha_i$ from a voter if she is placed at the $i^{th}$ position in the voter's preference order. The score of a candidate from a voter set is the sum of the scores she receives from each of the voters. If $\alpha_i$ is 1 for $i \in [k]$ and 0 otherwise, we get the $k$-approval rule. If $\alpha_i$ is 0 for $i \in [m-k]$ and $-1$ otherwise, we get the $k$-veto rule. The scoring rules for score vectors $(1, 0, \ldots, 0)$ and $(0, \ldots, 0, -1)$ are called plurality and veto rules respectively. The scoring rule for score vector $(m-1, m-2, \ldots, 1, 0)$ is known as the Borda rule.

The scores for the other voting rules studied in the paper (apart from scoring rules) are defined as follows. Let $vs(a, b)$ be the difference in the number of votes in which $a$ precedes $b$ and the number of votes in which $a$ succeeds $b$, for $a, b \in \mathcal{C}$. The maximin score of a candidate $a$ is $\min_{b\neq a} vs(a, b)$. The candidate with the maximum maximin score (after tie-breaking) is the winner. Given $\alpha \in [0, 1]$, the Copeland$^\alpha$ score of a candidate $a$ is $|\{b \neq a : vs(a, b) > 0\}| + \alpha \times |\{b \neq a : vs(a, b) = 0\}|$. The candidate with the maximum Copeland$^\alpha$ score (after tie-breaking) is the winner. We will assume $\alpha$ to be zero, if not mentioned separately. A candidate $a$ that has a positive pairwise score $vs(a, b)$ against all other candidates $b \in \mathcal{C}$ is called a Condorcet winner. Rules which select a Condorcet winner as the winner (whenever it exists) are called Condorcet-consistent rules. Copeland and maximin voting rules are Condorcet-consistent. The simplified Bucklin score of a given candidate $a$ is the minimum number $k$ such that more than half of the voters rank $a$ in their top $k$ positions. The candidate with the lowest simplified Bucklin score (after tie-breaking) is the winner and her score is called the Bucklin winning round.

We use $s(c)$ to denote the total score that a candidate $c \in \mathcal{C}$ gets in an election. Similarly, if $Y \subseteq \mathcal{C}$, we use $s(Y)$ to refer to the score of each candidate from $Y$; e.g. $s(Y) = 5$ means that each candidate from $Y$ has a score of 5. The voting rule under consideration will be clear from the context. Note that we assume that the briber is aware of the votes cast by each voter.

## 2.3 Our problem definition

We now define our problems formally. Let $r$ be a voting rule.

**Definition 2.1** (Effect of Applying Shift Vector on a Network of Voters)**.** Suppose we have a set $\mathcal{C}$ of $m$ candidates, a set $\mathcal{V}$ of $n$ voters, an $n$-voter profile $\mathcal{P} = (\succ_i^{\mathcal{P}})_{i\in[n]} \in \mathcal{L}(\mathcal{C})^n$ over $\mathcal{C}$, a directed weighted network $\mathcal{G} = (\mathcal{V}, \mathcal{A}, w : \mathcal{A} \longrightarrow \mathbb{R}^+)$, and a shift vector $\mathfrak{s} = (s_1, \ldots, s_n) \in \mathbb{N}_0^n$. We define a tuple $(s'_1, \ldots, s'_n) \in \mathbb{N}_0^n$ as follows.

$$s'_i = s_i + \sum_{j\in[n]\setminus\{i\}:(j,i)\in\mathcal{A}} s_j \cdot w(j,i)$$

Let $c \in \mathcal{C}$ be the candidate we want to make win. Let $\succ_i^{\mathcal{Q}}$ be the preference obtained from $\succ_i^{\mathcal{P}}$ by shifting $c$ to left by $\min\{s'_i, \text{pos}(c)\}$ positions. We call the profile $\mathcal{Q} = (\succ_i^{\mathcal{Q}})_{i\in[n]}$ to be the profile resulting from applying $\mathfrak{s}$ on $\mathcal{G}$.

**Problem Definition 1:** [Shift Bribery over Network ]
Given a set $\mathcal{C}$ of $m$ candidates, a set $\mathcal{V}$ of $n$ voters, an $n$-voter profile $\mathcal{P} = (\succ_i^{\mathcal{P}})_{i\in[n]} \in \mathcal{L}(\mathcal{C})^n$ over $\mathcal{C}$, a directed weighted network $\mathcal{G} = (\mathcal{V}, \mathcal{A}, w : \mathcal{A} \longrightarrow \mathbb{R}^+)$, the preferred candidate $c \in \mathcal{C}$, a preference $\succ_B \in \mathcal{L}(\mathcal{C})$ of the briber, a family $\Pi = (\pi_i : [m-1] \to \mathbb{N})_{i\in[n]}$ of cost functions (where $\pi_i(0) = 0, \forall i \in [n]$) corresponding to every voter, and a budget $b \in \mathbb{R}$, compute if there exists a shift vector $\mathfrak{s} = (s_1, \ldots, s_n) \in \mathbb{N}_0^n$ such that: (1) $\Sigma_{v_i\in\mathcal{V}}\pi_i(s_i) \leqslant b$ (2) the candidate $c$ is a winner in the profile which is obtained by applying the shift vector $\mathfrak{s}$ on $\mathcal{G}$. An arbitrary instance of Shift Bribery over Network is denoted by $(\mathcal{C}, \mathcal{P}, \mathcal{G}, c, \succ_B, \Pi, b)$.

Problem Definition 1 is the constructive version of the problem. We can also as well define and study the destructive version of the problem.

# Chapter 3

# Results

**definition:** Given an undirected graph $\mathcal{G} = (V, E)$, a subset of vertices D is called a dominating set if for every vertex $v \in V \backslash D$, there exists a vertex $u \in D$ such that $(u, v) \in E$.
An instance of DOMINATING SET PROBLEM $(\mathcal{G}, k)$ is a yes instance if there exists a dominating set of size $\leqslant k$, and no otherwise.

## 3.1 General graph network

**Theorem 3.1.** SHIFT BRIBERY OVER NETWORK *is* NP*-complete when all members of the family of cost functions* $\Pi$ *are identity functions and there are 2 candidates.*

*Proof:* Let $(\mathcal{G}, k)$ be an arbitrary instance of DOMINATING SET PROBLEM, consisting of n vertices. Let us construct an instance of SHIFT BRIBERY OVER NETWORK with 2 candidates and identity cost functions from this. Consider the vertices to be voters, and all the edges to have weight 1. Consider a set of candidates $\mathcal{C} = (0, 1)$, with 1 being our preferred candidate $c$. Let the preference profile of these n voters be [0, 1] (i.e they prefer candidate 0). Now, add n-1 isolated voters (vertices with no edges) with the same preference profile, thus generating a new graph $\mathcal{G}_1$, and a 2n-1-voter preference profile $\mathcal{P}$. The family of cost functions $\Pi$ is just the set of 2n-1 identity functions. The budget b is just the value k from the DOMINATING SET PROBLEM instance. $(\mathcal{C}, \mathcal{P}, \mathcal{G}_1, c, \succ_B, \Pi, k)$ is the instance of SHIFT BRIBERY OVER NETWORK

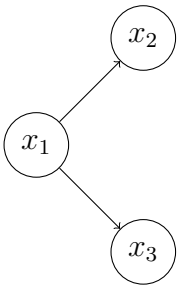


FIGURE 3.1: the graph $\mathcal{G}$ from the arbitrary instance of DOMINATING SET PROBLEM

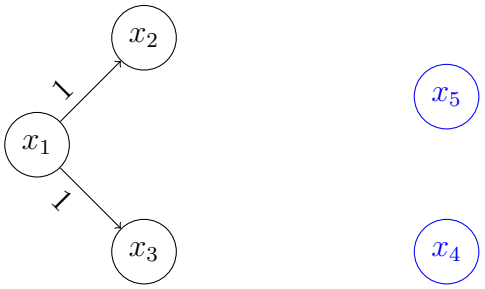


FIGURE 3.2: the new graph $\mathcal{G}_1$ with isolated voters added

It is easy to show that if the arbitrary instance of DOMINATING SET PROBLEM $(\mathcal{G}, k)$ is a 'yes' instance, then the constructed instance of SHIFT BRIBERY OVER NETWORK $(\mathcal{C}, \mathcal{P}, \mathcal{G}_1, c, \succ_B, \Pi, b)$ is also a 'yes' instance. Let us say $\mathcal{G}$ has a dominating set of size $\leqslant k$. Then the cost of bribing all the voters in this set is also $\leqslant k$, which is under the budget b (since b = k). Further, since the weights of the edges of the original graph $\mathcal{G}$ is 1, the shift vector of all n vertices in $\mathcal{G}$ is $\geqslant 1$ (since every vertex is a neighbour of a bribed vertex by definition of dominating set). Thus n voters have successfully changed vote from candidate 0 to to candidate 1, which is sufficient for our bribery scheme in SHIFT BRIBERY OVER NETWORK to be successful
Now to show that if the constructed instance of SHIFT BRIBERY OVER NETWORK is a 'yes' instance, then instance DOMINATING SET PROBLEM is also a 'yes' instance. Let us say we have a successful bribery scheme. Let us say we bribe a subset of voters $\mathcal{X}$ from $\mathcal{G}$, and the remaining voters $\mathcal{X}_1$ from $\mathcal{G}_1 \backslash \mathcal{G}$. If the voters we bribed is part of a dominating set $\mathcal{D}$ of $\mathcal{G}$, then we are done. Let $\bar{\mathcal{X}}$ be the set of vertices in $\mathcal{G}$ that either belong to $\mathcal{X}$ or are neighbours of vertices in $\mathcal{X}$. if $\mathcal{X}$ is not a dominating set of $\mathcal{G}$, then let the set L denote the non-empty set of vertices in $\mathcal{G}$ that aren't neighbours of vertices in $\mathcal{X}$.

Since the bribery scheme of SHIFT BRIBERY OVER NETWORK was successful, the number of voters voting for 1 must be $\geqslant n \implies |\bar{\mathcal{X}}| + |\mathcal{X}_1| \geqslant n$.
By definition, L and $\bar{\mathcal{X}}$ are complimentary sets of $\mathcal{G}$, and hence $|\bar{\mathcal{X}}| + |L| = n \implies |L| \leqslant |\mathcal{X}_1|$
Also, To satisfy budget constraints, we can bribe at most k voters, $|\mathcal{X}| + |\mathcal{X}_1| \leqslant k$.
Thus $|L| \leqslant |\mathcal{X}_1| \implies |\mathcal{X}| + |L| \leqslant k \implies |\mathcal{X} \cup L| \leqslant k$. Also, $\mathcal{X} \cup L$ is a dominating set of $\mathcal{G}$ by definition of L. Hence a dominating set of size $\leqslant k$ exists of $\mathcal{G}$ □

Note that by extension of the above theorem, the general version of SHIFT BRIBERY OVER NETWORK is also NP complete

## 3.2 Complete graph network

**Theorem 3.2.** SHIFT BRIBERY OVER NETWORK *is NP complete for complete graphs and 2 candidates*

*Proof:* Let $(\mathcal{G}, k)$ be an arbitrary instance of DOMINATING SET PROBLEM, consisting of n vertices. Let us construct an instance of SHIFT BRIBERY OVER NETWORK with 2 candidates and a new graph $\mathcal{G}_1$. Consider the vertices to be voters. Consider a set of candidates $\mathcal{C} = (0, 1)$, with 1 being our preferred candidate $c$. Let the preference profile of these n voters be [0, 1] (i.e they prefer candidate 0). Now, add n-1 voters (vertices) with the same preference profile [0, 1], thus completing our 2n-1-voter preference profile $\mathcal{P}$. Now consider all edges of $\mathcal{G}$ to have weight 1, and add edges of weight 1/2k between all pairs of unconnected vertices, thus generating a complete graph $\mathcal{G}_1$. The family of 2n-1 cost functions $\Pi$ is the set of n identity functions for the voters originally in $\mathcal{G}$, and the set of functions $\pi(s_i) = (k + 1) \times si$ for the n-1 added voters in $\mathcal{G}_1$. The budget b is just the value k from the DOMINATING SET PROBLEM instance. $(\mathcal{C}, \mathcal{P}, \mathcal{G}_1, c, \succ_B, \Pi, k)$ is the instance of SHIFT BRIBERY OVER NETWORK

It is easy to show that if the arbitrary instance of DOMINATING SET PROBLEM $(\mathcal{G}, k)$ is a 'yes' instance, then the constructed instance of SHIFT BRIBERY OVER

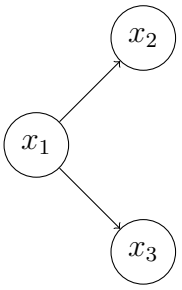


FIGURE 3.3: the graph $\mathcal{G}$ from the arbitrary instance of DOMINATING SET PROBLEM

NETWORK $(\mathcal{C}, \mathcal{P}, \mathcal{G}_1, c, \succ_B, \Pi, b)$ is also a 'yes' instance. It is the same as the proof for the previous theorem. Let us say $\mathcal{G}$ has a dominating set of size $\leqslant k$. Then the cost of bribing all the voters in this set is also $\leqslant k$, which is under the budget b (since b = k). Further, since the weights of the edges of the original graph $\mathcal{G}$ is 1, the shift vector of all n vertices in $\mathcal{G}$ is $\geqslant 1$ (since every vertex is a neighbour of a bribed vertex by definition of dominating set). Thus n voters have successfully changed vote from candidate 0 to to candidate 1, which is sufficient for our bribery scheme in SHIFT BRIBERY OVER NETWORK to be successful

Now to show that if the constructed instance of SHIFT BRIBERY OVER NETWORK is a 'yes' instance, then instance DOMINATING SET PROBLEM is also a 'yes' instance. Let us say we have a successful bribery scheme. Notice that it is not possible to bribe or change votes of voters from the added set of n-1 voters due to their very high cost functions. to get an shift effect of even 1 vote in any of these voters by bribing them, we need to spend $k + 1$, which is over budget. further, to get a shift effect in a voter of the added set by bribing its neighbours, we would need to spend $2 \times k$ on all neighbours combined, which is also over budget. Therefore, we can only bribe voters from the original graph $\mathcal{G}$. Let the set of voters bribed be $\mathcal{X}$. From budget constraints $|\mathcal{X}| \leqslant k$. If $\mathcal{X}$ is a dominating set of $\mathcal{G}$, then we are done. Let us assume its not, then there exists some voters in $\mathcal{G}$ whose votes have not changed. Also, it has already been established that the votes of the remaining n - 1 voters cannot change, meaning the total votes for our preferred candidate 1 would be $\leqslant n$. This contradicts our assumption that the bribery scheme was successful. Hence, $\mathcal{X}$ must be a dominating set of $\mathcal{G}$.

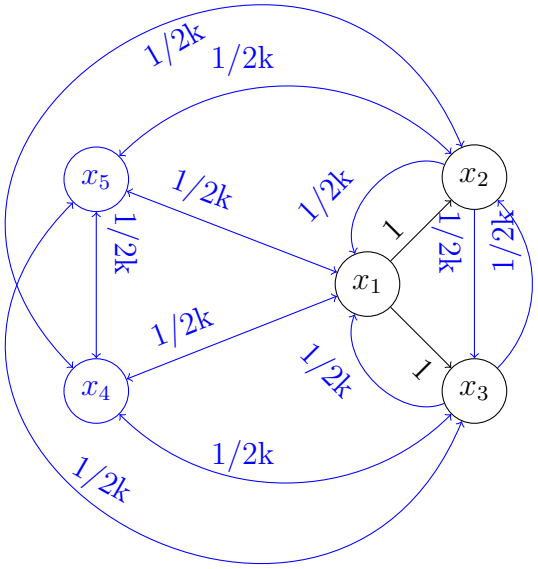


FIGURE 3.4: the new graph $\mathcal{G}_1$ with voters and 1/2k weighted edges added

□

**Theorem 3.3.** SHIFT BRIBERY OVER NETWORK *is polynomial-time solvable if the underlying graph is complete and all the edge weights are 1 for the majority voting rule, and linear cost functions.*

*Proof:* Let $(\mathcal{C}, \mathcal{P}, \mathcal{G}, c, \succ_B, \Pi, b)$ be an arbitrary instance of our problem, under the majority voting rule. Let us say the majority rule elects a dummy candidate in the event no majority is achieved. The goal of bribery is to then make our candidate c a Condorcet winner. The approach towards the solution is greedily solved. Notice that for all shift vectors $\mathfrak{s} = (s_i)_{i\in[n]}$, the value of $s'_i \ \forall i \in [n]$ is same and equal to $\sum_{i\in[n]} s_i$. Let us call this quantity $\mathcal{S}$.

Let us say the position vector of candidate c is $\mathfrak{p} = \left(\succ_i^{\mathcal{P}}(c)\right)_{i\in[n]}$ where $\succ_i^{\mathcal{P}}(c)$ specifies the position of desired candidate c in the ranking $\succ_i$ in preference profile $\mathcal{P}$. Notice that c is a condorcet winner $\iff \mathcal{S} \geqslant$ the median of this position vector. This is because all voters will see a shift of $\mathcal{S}$, and only those voters with $\succ_i^{\mathcal{P}}(c) \leqslant \mathcal{S}$ will vote for c under the plurality scoring rule.

Hence the minimum value of $\mathcal{S}$ needed to make c a condorcet winner can be easily found using any polynomial algorithm to find median in a vector. Let us say the family of linear cost functions for each voter are $\Pi = (\pi_i : [m-1] \to \mathbb{N})_{i\in[n]}$ (where $\pi_i(x) = b_i \times x, \forall i \in [n]$), and a budget $b \in \mathbb{R}$. This value of $\mathcal{S}$ can be achieved with minimum possible budget by bribing only the voter with the smallest cost coefficient, at a cost of $b_i \times \mathcal{S}$, (where $b_i$ is smallest among all $i \in [n]$). This is the minimum possible cost to make c a condorcet winner. If this cost is $\leqslant b$, then c can be made a condorcet winner. If not, c cannot be made a condorcet winner.

□

Let us now see the same problem with the plurality voting rule instead of the majority voting rule

**Lemma 3.4.** *define $\mathcal{S}$ as above. If c is a winner under the plurality voting rule for some $\mathcal{S}'$, then c is also a winner for all $\mathcal{S} \geqslant \mathcal{S}'$*

*Proof:* As discussed previously, all voters in this problem framework shift by the same maximum amount $\mathcal{S}$. Let us say c wins for a given value of $\mathcal{S}$. This means that c has more votes than all other candidates $c' \in \mathcal{C}$. We prove that this cannot change upon increasing $\mathcal{S}$. Notice that increasing the value of $\mathcal{S}$ cannot reduce the number of votes c gets, since all voters now shift c to a more preferable location. Similarly, Notice that increasing $\mathcal{S}$ cannot increase the number of votes a rival of c can get, since all other candidates either stay stationary, or shift to a less preferable profile. This completes the proof.

□

**Lemma 3.5.** *define $\mathcal{S}$ as above. If c is a winner under the plurality voting rule for some $\mathcal{S}'$, then c is also a winner for all $\mathcal{S} \leqslant \mathcal{S}'$*

*Proof:* Very analogous to the previous proof. Let us say c wins for a given value of $\mathcal{S}$. This means that c has less votes than some other candidate $c' \in \mathcal{C}$. We prove that this cannot change upon decreasing $\mathcal{S}$. Notice that decreasing the value of $\mathcal{S}$ cannot increase the number of votes c gets, since no voter will shift c to a more preferable location. Similarly, Notice that decreasing $\mathcal{S}$ cannot decrease the number of votes a rival of c can get, since all other candidates either stay stationary, or shift to a more preferable profile.

□

**Lemma 3.6.** *The optimal value of $\mathcal{S}$ has to be in the position vector $\mathfrak{p} = \left(\succ_i^{\mathcal{P}}(c)\right)_{i\in[n]}$ or 0.*

*Proof:* For simplicity, let us add 0 to our position vector Consider a value of $\mathcal{S}$ that is not equal to any $\succ_i^{\mathcal{P}}(c)$ for any $i \in [n]$ (or 0). Let us assume that this value of $\mathcal{S}$ is optimal (the lowest among all $\mathcal{S}$ that make c win). Now, consider the largest value of an element in our position vector that is smaller than $\mathcal{S}$. Let us call this element p. Notice that reducing the value of $\mathcal{S}$ to p cannot cause out candidate c to lose. This is because all voters with $\succ_i^{\mathcal{P}}(c)$ less than p voted for c previously, and will continue to do so. All the other voters didn't vote for c even with the old value of $\mathcal{S}$ since their $\succ_i^{\mathcal{P}}(c)$ was strictly greater than $\mathcal{S}$. This means that $p \geqslant \succ_i^{\mathcal{P}}(c) \iff \mathcal{S} \geqslant \succ_i^{\mathcal{P}}(c)$, which implies that the value p is also a valid $\mathcal{S}$ that can make c win. since $p < \mathcal{S}$, this contradicts the assumption that $\mathcal{S}$ is optimal.

Note that this proof does not work if we do not add 0 to our position vector, since $\mathcal{S} = 0$ could also be optimal (c is already the winner).

□

**Theorem 3.7.** Shift Bribery over Network *is polynomial-time solvable if the underlying graph is complete and all the edge weights are 1 for the plurality voting rule, and linear cost functions.*

*Proof:* for a given $\mathcal{S}$, we can simulate the vote and check if c is a winner in polynomial time. From **lemma 3.4** and **lemma 3.5**, it follows that we can binary search over possible values of $\mathcal{S}$ to find the smallest value of $\mathcal{S}$ that satisfies our agenda. From **lemma 3.6** it follows that we only need to check $\mathcal{S}$ values from our position vector $\mathfrak{p} = \left(\succ_i^{\mathcal{P}}(c)\right)_{i\in[n]} \cup [0]$. Hence, we need to search from a set of polynomial size, and checking each value of the set also takes polynomial time by generating new preference profiles for each voter. The time complexity to get the optimal value of $\mathcal{S}$ is therefore $O(n\log(n))$.

Once we have found the optimal value of $\mathcal{S}$, finding the optimal bribery scheme follows the same procedure as before.

The value $\mathcal{S}$ can be achieved with minimum possible budget by bribing only the voter with the smallest cost coefficient, at a cost of $b_i \times \mathcal{S}$. If this cost is $\leqslant b$, then c can be made a plurality winner, else not.

□

## 3.3 Path graph network

Consider paths with edge weights 1 and identical linear cost functions. without loss of generality, let us assume the $i^{th}$ voter to be connected to the $i+1^{th}$ by and edge of weight 1. consider that the shift vector $(s'_1,\ldots,s'_n) \in \mathbb{N}_0^n$. It now follows from our problem definition that $s'_i = s_i + s_{i+1}$. under plurality, for a voter i to vote for c, $s'_i \geqslant \succ_i^{\mathcal{P}}(c)$

**Lemma 3.8.** *There exists an optimal solution where* $s_i > \succ_i^{\mathcal{P}}(c) \Rightarrow s_i + s_{i-1} = \succ_{i-1}^{\mathcal{P}}(c)$

*Proof:* First consider the case where $s_i + s_{i-1}$ is greater than $\succ_{i-1}^{\mathcal{P}}(c)$. Now let us say there exists an optimal solution with $s_i > \succ_i^{\mathcal{P}}(c)$. We can decrease the value of $s_i$ until $s_i = max(\succ_i^{\mathcal{P}}(c), \succ_{i-1}^{\mathcal{P}}(c) - s_{i-1})$ without changing optimality. This is because we do not decrement $s_i$ past the point where either voter changes their vote. We have now generated a new optimal solution that satisfies the above property.

Now consider the case where $s_i + s_{i-1} <\succ_{i-1}^{\mathcal{P}} (c)$. Once again let us say there exists an optimal solution with $s_i >\succ_i^{\mathcal{P}} (c)$. We can decrease the value of $s_i$ to $\succ_i^{\mathcal{P}} (c)$ without changing optimality. Since the $i^{th}$ voter will continue voting for candidate c and the $i-1^{th}$ voter will continue to not vote for c. the value of $s_i$ does not impact any other voters.
This completes the proof

□

**Lemma 3.9.** *There exists an optimal solution where* $s_{i+1} > 0 \Rightarrow s_i + s_{i+1} =\succ_i^{\mathcal{P}} (c)$

*Proof:* let us say $s_{i+1} > 0$ and $s_i + s_{i+1} > \succ_i^{\mathcal{P}} (c)$. We can now reduce the value of $s_{i+1}$ to $max(0, \succ_i^{\mathcal{P}} (c) - s_i)$, while increasing $s_{i+2}$ by the same amount. Notice that this does not change optimality since we do not decrement $s_{i+1}$ enough to cause voter i to not vote for c. Also, since we incremented $s_{i+2}$, voter i+1 will also not change their vote (since $s_i + s_{i+1}$ has not changed). Subsequent voters can only be better off due to increasing $s_{i+2}$, since that increases the value of their shifts. Notice that this also does not change the optimality of cost, since all voters have the same cost function, and we are just transferring values of $s_i$
Now let us say $s_i + s_{i+1} < \succ_i^{\mathcal{P}} (c)$. We can reduce $s_{i+1}$ to 0 and increment $s_{i+2}$ by the same amount. This also yields an optimal solution since it neither changes voter i's vote, nor voter i+1. Subsequent voters can only be better off due to increasing $s_{i+2}$, since that increases the value of their shifts. Once again, this does not affect the optimality of costs, since all voters have identical cost functions.

□

**Lemma 3.10.** *There exists an optimal solution where* $s_0$ *is 0.*

*Proof:* Similar logic to above, transfer $s_0$ to $s_1$. This does not affect $\succ_0^{\mathcal{P}} (c)$, and can only positively affect subsequent voters.

□

From the above 3 lemmas, it follows that for every value $s_{i+1}$ we only need to worry about satisfying $\succ_i^{\mathcal{P}}(c)$. for any $s_{i+1}$, we can either choose to satisfy the $i^{th}$ voter's shift requirement, or disregard it completely and set $s_{i+1}$ to 0. Also since all cost functions are identical and linear, we only need to worry about minimising $\sum_{i\in[n]} s_i$ For the majority voting rule this gives us a dynamic programming solution as follows:

*Algorithm*: define $dp[val][i][k]$ to be the minimum budget used until voter i (not included) such that there are k more voters left to achieve majority, and the value of $s_i$ is val. Initialise $dp[p0][1][ceil(n/2)-1]$ to $\succ_0^{\mathcal{P}}(c)$ (to satisfy the first voter with $s_1$). Initialise all other values of dp to -1. At each step we can either chose to satisfy another voter, or not:

If we choose to satisfy voter i:

$$if(val < \succ_i^{\mathcal{P}}(c)) \text{ then } dp[\succ_i^{\mathcal{P}}(c) - val][i+1][k-1] = max(dp[\succ_i^{\mathcal{P}}(c) - val][i+1][k-1], dp[val][i][k] + \succ_i^{\mathcal{P}}(c) - val)$$

$$if(val > \succ_i^{\mathcal{P}}(c)) \text{ (voter i already satisfied) then } dp[0][i+1][k-1] = max(dp[0][i+1][k-1], dp[val][i][k])$$

If we choose not to satisfy voter i: (val must be less than pi)

$$dp[0][i+1][k] = max(dp[0][i+1][k], dp[val][i][k])$$

Note that from **Lemma 3.9**, the maximum possible value of any val is $max(\succ_i^{\mathcal{P}}(c))i \in [n]$. This quantity cannot be more than n. Hence the number of dp states is at most $n \times n \times ceil(n/2)$. Solving the dp either recursively or iteratively will lead to a polynomial time solution, since states are not revisited.
Note: There might be more space or time optimised solutions to this problem. This algorithm is simply to show that the problem is not NP-Hard

## 3.4 Tree network

**Theorem 3.11.** Shift Bribery over Network *is polynomial-time solvable if the underlying graph is a tree with undirected edges, there are only 2 candidates, all edge weights are 1, and all cost functions are identical and linear*

**Algorithm.** Since all cost functions are identical and linear, There is a simple upper limit on the number of voters we can bribe, which is given by $b/k$, where k is the linear constant on the cost function. Lets call this max value T.
consider a tree of n voters with directed edges. Let dp1[i][b][t] be the maximum number of voters we can get to vote for our candidate in the subtree rooted at node i with at most t voters bribed. If b = 0, voter i isn't bribed. If b = 1, voter i is bribed. Notice that since there are only 2 candidates with edge weights 1, it provides no advantage to bribe a voter by more than 1.
Let dp2[i][b][t] be the same quantity as above, but without counting the root of the subtree. This will be used for transitions and computing it from the previous dp is non trivial, since we need to go through children and check if they are bribed. Hence, it is stored separately. Clearly, if we can solve the dp, we can solve the problem by checking if

$$max(dp1[root][1][T], dp1[root][0][T] \geqslant n/2.$$

To calculate the the dp, let us define a knapsack function $K(V_1, V_2, V_3, ...V_m)[t]$ as $\max_{t_1+t_2+t_3+...+t_m=t} (\sum_{i=1}^{m} V_i[t_i])$, where $V_i$ are given functions. The intuition behind this function is to compute the maximum number of voters we can get from m children of a node with at most t voters bribed. Also notice that this function can be calculated iteratively given $V_i$ by simply merging the subproblems 2 sets at a time as follows: $K(V_1, V_2, ...V_m) = K(K(V_1, V_2, ....V_{m-1}, V_m)$. Also, merging any 2 sets takes maximum $\mathcal{O}(t)$ time by iterative over all possibilities of t1+t2 = t.
Let Ci be the set of children of node i, and Cij be the jth child of i. Also, define P[i] to be 1 if i previously voted for our candidate, or 0 otherwise. Then
Let

$$V_j[t] = \max(dp1[C_{ij}][0][t], dp1[C_{ij}][1][t])$$

$$\text{set } flag[j][t] = flag[j][t] \vee 1 \text{ if } V_j[t] = dp[C_{ij}][1][t], \text{ else } flag[j][t] = flag[j][t] \vee 0$$
$$V'_j[t] = \max(dp2[C_{ij}][0][t], dp2[C_{ij}][1][t])$$

At every node, compute the function $K(V_1, \ldots, V_{|C_i|})[t]$ and $K(V'_1, \ldots, V'_{|C_i|})[t-1]$. While computing $K(V_1, \ldots, V_{|C_i|})[t]$, compute $flag = flag[i][t_i] \vee flag \ \forall i \in Ci$, where $t_i$ is the value selected in the knapsack function $K(V_1, V_2, V_3, ...V_m)[t] = \max_{t_1+t_2+t_3+...+t_m=t} (\sum_{i=1}^{m} V_i[t_i])$. The purpose of the flag is to check if any of the children of node i have been bribed directly. Now:

$$\text{dp2}[i][0][t] = K(V_1, \ldots, V_{|C_i|})[t]$$
$$\text{dp1}[i][0][t] = \text{dp2}[i][0][t] + max(P[i], flag)$$
$$\text{dp2}[i][1][t] = |C_i| + K(V'_1, \ldots, V'_{|C_i|})[t-1]$$
$$\text{dp1}[i][1][t] = \text{dp2}[i][1][t] + 1$$

**Base case:** At a leaf node i, initialise

$$\text{dp1[i][0][t] = P[i], dp2[i][0][t] = 0}$$
$$\text{p1[i][1][t1] = 1, dp2[i][1][t1] = 0 where t1} \geqslant 1$$

*Proof:* Let us show that if the dp states hold for the children of a node for all values of t, then they also hold for the parent. Firstly, it is trivial to verify that the dp states are correctly initialised for leaf nodes. As mentioned before, let Ci be the set of children of node i, and Cij be the jth child of i. Define P[i] to be 1 if i previously voted for our candidate, or 0 otherwise.

$V_j[t] = \max(dp1[C_{ij}][0][t], dp1[C_{ij}][1][t])$ Here $V_j[t]$ represents the maximum possible answer in the subtree at $C_{ij}$ regardless of the rest of the tree. Considering no bribery to i, all voters currently voting for our candidate in the subtree rooted at i (not including the root node itself for simplicity), is simply the sum of all voters voting in the subtrees at the children. It then follows that the maximum possible voters voting for our candidate given t voters bribed is the maximal sum of voters in the individual subtrees with total bribed voters = t. By the definition of our knapsack function, this yields the following recursive relation:

$$\text{dp2}[i][0][t] = K(V_1, \ldots, V_{|C_i|})[t]$$

Now, computing the answer in the subtree rooted at i including i is a little more complicated. If i already votes for our desired candidate (i.e. $\mathrm{P}(C_i) = 1$), then the answer is simply $\mathrm{dp2}[i][0][t] + 1$ in all cases. But suppose i doesn't already vote for our desired candidate. Notice that choosing a sub optimal maximal sum $\leqslant$ (optimal sum - 1) can never yield an optimal result for the subtree at i since the node i itself can only add 1 voter's value. Hence, we do not need to change the results of the knapsack function. Simply check in the current max formulation of the knapsack function if any of the children were bribed in the maximal sum through the flag variable as given. If yes, then its effect will cause i to vote for our desired candidate after bribery. Also, whenever $dp1[C_{ij}][0][t]$ and $dp1[C_{ij}][1][t]$ are the same, the flag is set as though $dp1[C_{ij}][1][t]$ has been selected. This is because the root of the subtree being bribed cannot be detrimental to the remaining tree. This proves the relation

$$\mathrm{dp1}[i][0][t] = \mathrm{dp2}[i][0][t] + max(P[i], flag)$$

Coming to the cases where the current node i is bribed. Since i is bribed, all of its children would automatically vote for our desired candidate. Hence, the answer in the subtree rooted at i is just [1 (for node i)] + [$|C_i|$ (for the children of i)] + [sum of number of voters in subtrees at children, not including the children themselves]. Therefore, the maximal answer for any t is given by the [maximal sum of voters in the subtrees adding to (t-1)] + [1] + [$|C_i|$]. By definition of the knapsack function, the maximal sum can be given by

$$K(V'_1, \ldots, V'_{|C_i|})[t-1] \text{ where } V'_j[t] = \max(dp2[C_{ij}][0][t], dp2[C_{ij}][1][t])$$
$$\text{Hence } \mathrm{dp2}[i][1][t] = |C_i| + K(V'_1, \ldots, V'_{|C_i|})[t-1]$$
$$\mathrm{dp1}[i][1][t] = \mathrm{dp2}[i][1][t] + 1$$

□

**time complexity analysis**: Note that $K(K(V_1, \ldots, V_{m-1}), V_m) = K(V_1, \ldots, V_m)$, and that $K(A, B)$ can be directly calculated by the above formula in $O(|A| \cdot |B|)$ time. Hence calculating $K(V_1, \ldots, V_m)$ takes $O(\sum_{i=2}^{m} |V_i| \sum_{i-1}^{j=1} |V_j|)$ time regardless of the order we combine the sets in.

Calculating the answer with this recursion takes $O(nT)$ time. Informally, this is because over the course of the algorithm, we combine single-element DPs into a DP representing the whole tree. Consider a worst case merging scenario merging sets of size T. We do at most $n/T$ combinations of sets of size $T$, and any element costs us at most $2T$ time (if element $x \in A$ costs us $|B|$ time when calculating $K(A, B)$) before it gets merged into a set of size $T$, so the total amount of work is at most $O(T^2 n/T + Tn) = O(nT)$.

Note that Knapsack function computation would be polynomial in sum of costs in case of non identical linear cost functions

**Theorem 3.12.** Shift Bribery over Network *is polynomial-time solvable if the underlying graph is a tree with directed edges, there are only 2 candidates, all edge weights are 1, and all cost functions are identical and linear*

*Proof:* The only difference in the algorithm in both cases is that for directed edges, we no longer need to consider the effect of children on its parent when the parent isn't bribed. Hence, we no longer need a flag variable and the transitions become as follows:

$$V_j[t] = \max(dp1[C_{ij}][0][t], dp1[C_{ij}][1][t])$$
$$V'_j[t] = \max(dp2[C_{ij}][0][t], dp2[C_{ij}][1][t])$$

$$\mathrm{dp2}[i][0][t] = K(V_1, \ldots, V_{|C_i|})[t]$$
$$\mathrm{dp1}[i][0][t] = \mathrm{dp2}[i][0][t] + P[i]$$
$$\mathrm{dp2}[i][1][t] = |C_i| + K(V'_1, \ldots, V'_{|C_i|})[t-1]$$
$$\mathrm{dp1}[i][1][t] = \mathrm{dp2}[i][1][t] + 1$$

□

# Chapter 4

# Future work and conclusions

## 4.1 Future work

We are yet to find a polynomial algorithm for trees with multiple candidates and arbitrary weights.

Further, the previous algorithm for trees needs to be augmented to work on graphs with some fixed tree width

## 4.2 Conclusions

Electoral manipulation is an important topic with a lot of real life implications in both human systems and systems with computer agents. There has been a lot of work on the complexity of electoral manipulation either through control or bribery schemes. We modelled the bribery problem considering the influence of voters on their network. This particular model to the bribery problem has not been studied in previous works. Our problem definition attempts to bridge that gap, while providing new insights into the nature and vulnerability of various electoral systems in the social, inter-connected world.

We found the complexity class of many versions of the problem, along with useful algorithms where possible. However, there is still work to be done for generalised tree structures and real world graphs that can be modelled with a small tree width.

# Bibliography


Bartholdi, J., Tovey, C., and Trick, M. (1989). The computational difficulty of manipulating an election. *Soc. Choice Welf.*, 6(3):227–241.

Bartholdi III, J. J., Tovey, C. A., and Trick, M. A. (1992). How hard is it to control an election? *Math Comput Model*, 16(8-9):27–40.

Bentert, M., Koana, T., and Niedermeier, R. (2021). The complexity of gerrymandering over graphs: Paths and trees. *arXiv preprint arXiv:2102.08905*.

Bredereck, R., Faliszewski, P., Niedermeier, R., and Talmon, N. (2016). Large-scale election campaigns: Combinatorial shift bribery. *J. Artif. Intell. Res.*, 55:603–652.

Chakraborty, A., Patro, G. K., Ganguly, N., Gummadi, K. P., and Loiseau, P. (2019). Equality of voice: Towards fair representation in crowdsourced top-k recommendations. In *Proc. 2nd ACM Conference on Fairness, Accountability, and Transparency, FAT*, pages 129–138.

Cohen-Zemach, A., Lewenberg, Y., and Rosenschein, J. S. (2018). Gerrymandering over graphs. In *Proc. 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS*, pages 274–282.

Conitzer, V., Lang, J., and Sandholm, T. (2003). How many candidates are needed to make elections hard to manipulate? In *Proceedings of the 9th Conference on Theoretical Aspects of Rationality and Knowledge*, pages 201–214.

Conitzer, V. and Sandholm, T. (2002). Complexity of manipulating elections with few candidates. In *AAAI/IAAI*, pages 314–319.

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

Elkind, E. and Faliszewski, P. (2010). Approximation algorithms for campaign management. In *Internet and Network Economics - 6th International Workshop, WINE*, pages 473–482.

Ephrati, E. and Rosenschein, J. S. (1991). The clarke tax as a consensus mechanism among automated agents. In *AAAI Conference on Artificial Intelligence*.

F. Brandt, V. Conitzer, U. E. J. L. and Procaccia, A. (2015). *Handbook of computational social choice*, volume 1. freeman New York.

Faliszewski, P., Hemaspaandra, E., and Hemaspaandra, L. A. (2009). How hard is bribery in elections? *J. Artif. Intell. Res.*, 35(2):485–532.

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

Hemaspaandra, E., Hemaspaandra, L. A., and Rothe, J. (2007). Anyone but him: The complexity of precluding an alternative. *Artif. Intell.*, 171(5-6):255–285.

Ito, T., Kamiyama, N., Kobayashi, Y., and Okamoto, Y. (2019). Algorithms for gerrymandering over graphs. In *Proc. 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 1413–1421.

Lev, O. and Lewenberg, Y. (2019). Reverse gerrymandering: Manipulation in multigroup decision making. In *Proc. 33rd Conference on Artificial Intelligence, AAAI*, pages 2069–2076.

Pennock, D. M., Horvitz, E., and Giles, C. L. (2000). Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering. In *AAAI/IAAI*.

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

Taylor, A. D. (2005). Social choice and the mathematics of manipulation.