COMPUTATIONAL SOCIAL CHOICE THEORY: THE CLASS $\mathbf{P}_{||}^{\mathbf{NP}}$, WINNER DETERMINATION AND MANIPULATION

A dissertation submitted to Birkbeck, University of London for the degree of MSc in Mathematics.

By
Konstantinos Bampalis
Supervisor: Dr. Maura Paterson
Department of Economics, Mathematics and Statistics

30st August 2024

Contents

Acknowledgments

I want to thank my supervisor Dr. Maura Paterson for all her help and support, not only throughout this dissertation but also throughout my MSc and my time at Birkbeck. Her eagerness to assist with anything, Mathematics or otherwise, has been invaluable.

I also want to thank all the people in the Mathematics department for creating a great learning and supportive environment.

Finally, to my parents, brother and friends for their constant support and encouragement, a very big thank you!

Abstract

BIRKBECK, UNIVERSITY OF LONDON

ABSTRACT OF DISSERTATION submitted by **Konstantinos Bampalis** and entitled **Computational Social Choice Theory: The Class $\mathbf{P}_{||}^{\mathbf{NP}}$, Winner Determination and Manipulation**.

Date of Submission: 30st August 2024

In this dissertation, we study the Computational Complexity of problems related to Voting.

Declaration

Chapter 1

Preliminaries

1.1 Social Choice Theory

Social Choice Theory studies the design and analysis of collective decision-making procedures. Its main questions centre around how can groups of agents with competing preferences choose an outcome from a set of alternatives, what are the different methods for doing so and what are the properties of different methods. In essence, Social Choice Theory is concerned with voting scenarios. The following is mostly from [25], [1] and.

Formally, we have a set $N = \{1, ..., n\}$ of voters and a set A, $|A| = m$, of alternatives. Voters are represented by their preferences over the alternatives, which is a ranking[1] of A. We let P_i denote the preferences of voter i.

Example 1.1.1. Let $N = \{1, 2\}$ and $A = \{a, b, c\}$. $P_1 = [b, a, c], P_1 = [a, c, b]$.

Let $\mathcal{L}^n$ denote all possible preferences over A. A preference profile, P, is an element of $\mathcal{L}^n$, where $P = \{P_1, ..., P_n\}$. A Voting Mechanism is a rule that dictates how to pick the winner of an election. This is nothing more than a map, $\mathcal{E} : \mathcal{L}^n \to 2^A$.

We now consider the main voting rules that will be used throughout the dissertation. We consider voting rules based on Scoring Protocols, Pairwise Comparisons and Sequential Voting.

Definition 1.1.2. A Scoring Voting Protocol is defined by a vector $\alpha \in \mathbb{R}^m$. Each voter $i \in N$ gives α_j points to the candidate they ranked at the jth position and $\max\{\alpha_p\}$ from α wins.

[1]A complete, transitive and asymmetric binary relation

Several rules exist based on this idea. Their only difference is in the allocation of the points.

Example 1.1.3 (Plurality Rule)**.** In the Plurality Rule, $\alpha_j = 1$ if $j = 1$. Otherwise, it is 0.

Example 1.1.4 (k-Approval Rule)**.** In the k-Approval Rule, $\alpha_j = 1$ for a $j = 1, ..., k$ for a fixed $k \leq m$. Otherwise, it is 0.

Example 1.1.5 (Borda Rule)**.** In the Borda Count, $\alpha_j = m - j$.

Definition 1.1.6. In Pairwise Comparison rules, candidates do not receive points immediately. Instead, all candidates engage with each other in a head-to-head contest and it is tested who is preferred by all voters using a majority criterion.

Example 1.1.7 (Condorcet Rule)**.** In the Condorcet Rule, the winner is the candidate who is preferred to each other candidate in a pairwise comparison by more than half of the voters.

Example 1.1.8 (Copeland Rule)**.** In the Copeland Rule, all candidates enter a head-to-head contest. Points are determined as follows; If one of the two candidates is preferred by a majority, that candidate receives one point and the other zero. Else, they receive half a point each. The winner is the candidate with the most points.

Example 1.1.9 (Maximin Rule)**.** In the Maximin Rule, for candidates $a, b \in A$, we let $p(a, b)$ be the number of voters that prefer a over b, For every a, we take the minimal value p over $A - \{a\}$. The rule chooses the candidate for which that value is maximal.

Definition 1.1.10. In Sequential Voting, voting proceeds in several stages, where in each stage, only a subset of the candidates or parts of the preferences of each voter are considered. The next stage depends on the result of the previous stage.

Example 1.1.11 (Single Transferable Voting)**.** In the STV Protocol, we proceed in rounds, given, in a one-off manner the votes on A. The number of rounds is not fixed, but it is at most the number of candidates. In each round, the candidate in the top position of a voter receives one point. If there is a candidate that receives a point from more than half of the voters, he wins. Otherwise, we delete the candidate with the least votes and redistribute the implicit votes. This is repeated until a candidate scores the majority.

The fact that several voting rules exist, yields the natural question as to which system is the best. There is no correct answer, nor a universally best voting system. Instead, we look at various properties that seem desirable and combinations thereof. One of the most interesting parts of Social Choice Theory is that certain combinations of properties are inconsistent. Many voting systems cannot satisfy certain properties simultaneously. There are many properties and many Impossibility Results, with Arrow's Theorem being the most celebrated one, however, relevant to this dissertation are the following.

Definition 1.1.12 (Monotonicity)**.** A Voting Rule $\mathcal{E}$ is monotonic if for every preference profile P and candidate x, the following hold:

i. If x is the winner, and

ii. if a new profile $P^{'}$, generated by P, where $P^{'}$ improves the position of x in some of the P_js but leaves everything else unchanged

Then, $P^{'}$ is still the winner.

Definition 1.1.13 (Responsiveness)**.** Let $s : \mathcal{L} \times A \rightarrow \mathbb{R}$ be a function. A Voting Rule $\mathcal{E}$ is Responsive if for every $P \in \mathcal{L}$, $\mathcal{E}$ chooses the candidate maximizing $s(P, x)$.

Definition 1.1.14 (Non-Dictatorship)**.** A Voting Rule $\mathcal{E}$ is Dictatorial if there is a voter $v \in N$ such that for every election, the outcome $\mathcal{E}(P)$ coincides with the preference profile of v, P_v, regardless of the remaining preference lists, $P - P_v$.

Definition 1.1.15 (Resoluteness)**.** A Voting Rule $\mathcal{E}$ is Resolute if $|\mathcal{E}(P)| = 1$

Definition 1.1.16 (Sovereignty)**.** A Voting Rule $\mathcal{E}$ satisfies Sovereignty if the map is surjective.

Definition 1.1.17 (Strategy-Proofness)**.** A Voting Rule $\mathcal{E}$ is Strategy-Proof if no voter can benefit from reporting an untruthful vote. If a Voting Rule is not Strategy-Proof, it is called manipulable.

Example 1.1.18. Let $N = \{1, 2, 3\}$ and $A = \{a, b, c, d\}$ with $P_1 = P_2 = [b, a, c, d]$ and the sincere preferences of voter 3, $P_3 = [a, b, c, d]$. Under the Borda Rule, candidate b is the winner with eight points. However, voter 3 wants candidate a to win. By having complete information of the preferences of all other voters and by reporting an insincere preference profile, such as $P_3 = [a, c, d, b]$, then can make b the winner. We see that the Borda Rule is manipulable.

Ideally, we want to be able to design strategy-proof voting rules. Unfortunately, this is not possible.

Theorem 1.1.19 (Gibbard-Satterthwaite)**.** *If there are at least* 3 *candidates, there is no preference-based voting system that simultaneously satisfies Non-Dictatorship, Resoluteness, Sovereignty and Strategy-Proofness.*

Proof. Omitted due to space. The interested reader is referred to [15], [27]. A concise version can be found in [7] □

Observe what the implications of Theorem 1.1.19 are. In practice, most of our voting systems yield a single winner, are not dictatorships and do not exclude any candidate from winning. It follows that all of our voting rules are susceptible to strategic voting and, thus manipulable.

1.2 Computational Complexity Theory

Computational Complexity Theory studies the resources required to solve computational problems. Its main questions centre around: How to classify problems in terms of similar difficulty, What are the relationships between these classes and how to formally prove these relationships [12]. The setup is as follows; Most of these can be found in: [3],[28],[22], [16] and [31].

Definition 1.2.1. An alphabet Σ is a finite set of symbols. A string over an alphabet is a finite sequence of symbols from Σ. The set Σ^* denotes Σ^*. A Language L is a subset of Σ^*.

Throughout this dissertation, we take $\Sigma = \{0, 1\}$, which is the binary alphabet. We defined alphabets, strings and languages to formally define computational problems. A computational problem is a question to which we would like the answer to. It is standard in Complexity Theory to restrict ourselves to Decision Problems; These are problems where given some input x, we answer yes or no, depending on whether x satisfies some predicate P. On the input side, every finite object or structure, (integers, tuples, matrices, graphs), is encoded as a string to be given as input to our machine model. On the output side, yes is represented by 1 and no by 0. Under this setup, we study Boolean Functions and we call decision problems, languages.

Definition 1.2.2 (Problems)**.** Let $f : \Sigma^* \to \Sigma$. Define as a decision problem or language the set $L_f = \{x : f(x) = 1\}$.

That is, this is the set for which the answer to the Decision Problem is yes. We solve computational problems using algorithms; A set of fixed rules that can be carried out mechanically. This is formalized with our model of computation, the Turing Machine.

Definition 1.2.3 (Turing Machine)**.** A Turing Machine, M, is a $7-$tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$, where Q is a set of states, Σ is the input alphabet, Γ is the tape alphabet, $\delta : Q \times \Gamma \times \{Left, Right\}$ is the transition function and q_0, q_a, q_r are the initial, accept - halt and reject - halt states.

Using a Turing Machine to actually compute something tends to be too tedious. Thus it suffices to understand the following: For any language L, if we have a Turing Machine that halts in the accept state whenever $x \in L$, and rejects otherwise, then we say that the Turing Machine decides L and call the language decidable.

To classify problems in terms of their difficulty we need to identify the amount of computational resources that they require. One such resource is Time, whereby time, we mean the number of steps that a Turing Machine takes until it halts.

Definition 1.2.4 (Running Time)**.** Let M be a deterministic Turing Machine that halts on all of its inputs. The running time of M is a function $t : \mathbb{N} \to \mathbb{N}$, where $t(n)$ denotes the number of steps that M takes until it halts, on any input of size n.

In practice, analyzing the running time of a Turing Machine, exactly, may yield an 'ugly' or too complicated function. For this reason, we analyze running asymptotically, ignoring lower-order terms. The following definition and example is from [28].

Definition 1.2.5 (Big O Notation)**.** Let f, g be functions, $f, g : \mathbb{N} \leftarrow \mathbf{N}$. We have $f(n) \in O(g(n))$ if $\exists c \in \mathbb{R}, n_0 \in \mathbb{N}$ such that $\forall n > n_0 f(n) \leq g(n)$.

Example 1.2.6. Let $f(n) = 5n^3 + 2n^2 + 22n + 6$. The highest order term is $5n^3$ and by disregarding the coefficient we have that $f(n) \in O(n^3)$. More formally, set $c = 6, n_0 = 10$ and then $f(n) \leq 6n^3$ for every $n \geq n_0$.

We are now going to look at some of the most important complexity classes. Intuitively, a complexity class is a set of similarly hard languages.

Definition 1.2.7 (Time)**.** Let $t : \mathbb{N} \to \mathbb{N}$ and M be Deterministic Turing Machine. The class $\mathbf{TIME(t(n))}$, is the set of languages decidable in $O(t(n))$ time.

Definition 1.2.8 (The Class **P**)**.** The class **P** is the set of languages decidable in polynomial time using a deterministic Turing Machine. Formally, $\mathbf{P} = \bigcup_k \text{TIME}(n^k)$

The class **P** is of central importance in Computational Complexity Theory as it represents efficiently solvable problems. As an illustrative example, consider the problem of determining whether there is an $s - t$ path in a directed graph G. This example, apart from simply showing membership into the class **P**, is important as it illustrates the concept of efficient computation. We call this problem `DIRECTED PATH`.

Example 1.2.9. Let $G = (V, E)$ be a directed graph where V is the set of vertices and E is the set of edges. Let $s, t \in V$ and suppose we want to determine whether there exists a path from vertex s to vertex t. Before we give an efficient algorithm, consider the brute-force approach; Given G enumerate all possible paths in G. Clearly, this is exponential in the size of vertices (approximately, the number of potential paths is n^n, where n is the number of vertices) and hence impractical for large graphs. Instead, consider the following method; Mark successively all vertices that can be reached from the starting vertex s. The vertices are reached by exploring all possible neighbours at each depth. Clearly, this is polynomial in the number of vertices. More formally, we describe the algorithm below and show that `DIRECTED PATH` $\in \mathbf{P}$.

Algorithm 1: `Breadth-FirstSearch`

```
Data: $G$, $s$, $t$
Result: Path between $s$ and $t$
Mark vertex $s$;
while additional vertices are being marked do
    Scan edges of $G$;
    for each edge $(x, y)$ in $G$ do
        if $x$ is marked and $y$ is unmarked then
            Mark $y$;
        end
    end
end
if $t$ is marked then
    return $s$-$t$ path exists;
else
    return $\emptyset$;
end
```

The proof that this algorithm runs in polynomial time and hence `DIRECTED PATH` $\in$ **P**, is easy and shown below

Proof. The first and last parts of the algorithm are executed only once, hence it suffices to concentrate on the loop which will run at most $|V| = n$ times. Inside the loop, we simply test, hence, the algorithm is polynomial in the size of the input and thus we have a polynomial time algorithm for the `DIRECTED PATH`. problem. □

The main point that this problem illustrates is that there was some sort of clever trick that allowed us to avoid a brute-force approach. However, for many problems, such attempts have failed and a polynomial algorithm has not been found (yet?). Furthermore, attempts to show that a polynomial algorithm need not exist have also failed. Whether we have not been clever enough or whether these problems are inherently difficult is one of the most important questions in Mathematics and Theoretical Computer Science. However, such problems have a very important feature, highlighted in the following example[2].

Example 1.2.10. Suppose we are given n cities and let $d_{i,j}$ represent the distance between

[2]This example is taken from my essay on the Mathematics Essay module, completed in 2023. It is taken from [22]

any two cities i, j, with $d_{i,j} = d_{j,i}$. A tour of the cities, involves starting from some city x, visiting all other cities and returning back to x and the cost of the tour is the sum of all distances incurred. Let B denote a target cost; Is there a tour of cost at most B? This is known as the `TRAVELLING SALESPERSON` problem and obviously the naive solution is to enumerate all possible tours, calculate their cost and then answer whether the cost condition is satisfied. This is $O(n!)$, which is clearly not polynomial and so far no polynomial time algorithm has been found. However, this problem has a very interesting feature; Assume that we are magically given the solution. Then, we can efficiently verify it. How? Check if the solution is indeed a tour, i.e. visits all the cities and then check if the cost is at most B, a polynomial - time procedure.

This example highlights a very important feature of computational problems; Efficiently Solving vs. Efficiently Verifying. We now consider another important class of problems, the class **NP** which includes the problems whose solutions can be verified in polynomial time. Clearly, $\mathbf{P} \subseteq \mathbf{NP}$ since to verify the solution, we can simply solve the problem. More formally, we have:

Definition 1.2.11. Let A be language, let V be an algorithm, the verifier and let c be the certificate. A verifier for A is an algorithm such that $A = \{w : V accepts \langle w, c \rangle for c\}$

Definition 1.2.12. The class **NP** is the class of languages that have Polynomial Time Verifiers.

Example 1.2.13. Let $n \in \mathbf{Z}$ Consider the problem of determining whether n has a factor in a given interval. Call this problem `FACTORING`. For a verifier, we can use a factorization of n, hence `FACTORING` $\in$ **NP**.

Example 1.2.14. More generally, let L be any problem that is in **P**. Design a verification algorithm for L by ignoring the certificate and check if an input x in L. Since this can be done in polynomial time, the verifier works in polynomial time hence L is also in **NP**.

Alternatively, we can define the class **NP** using Non-Deterministic Turing Machines. A Non-Deterministic Turing Machine is similar to a (Deterministic) Turing Machine, however, at each stage, instead of moving to the next state based on a deterministic transition function, it follows several possibilities based on a transition relation. The intuitive way to think about Non-Deterministic Turing Machines is to consider them as a parallel computing device with more power than standard Turing Machines. It is important to highlight that Non - Non-

Deterministic Turing Machines do not yield any randomness; Given an input, an NDTM will produce the same output on each run. We do not consider these machines anymore yet, define $mathbf NP$ for the sake of completeness.

Definition 1.2.15. The class **NP** is the set of languages that can be computed in polynomial time using a Non-Deterministic Turing Machine.

Within **NP**, there exist certain problems whose complexity relates to the entire class. Very informally, these problems are the same, in the sense that if X and Y are such problems we can convert X into a Y, and conversely. More formally, a polynomial (or lack thereof), algorithm for one of these problems implies a polynomial time for all such problems. These problems are called $\mathbf{NP}-$Complete.

Definition 1.2.16. Let A and B be languages. A is polynomially time reducible to B, $A \leq_P B$ if there exists a polynomially computable function $f : \Sigma^* \leftarrow \Sigma^*$ such that for all w, $w \in A \iff f(w) \in B$.

Definition 1.2.17. A problem B is $\mathbf{NP}-$Complete if:

i. $B \in \mathbf{NP}$

ii. $\forall A \in \mathbf{NP}$, $A \leq_P B$

Once we are given an $\mathbf{NP}-$Complete problem, the strategy is straightforward; Obtain others by means of a reduction. However, whether an $\mathbf{NP}-$Complete should exist in the first place, is not trivial. The first $\mathbf{NP}-$Complete problem was established by Cook and Levin in the 70s and is the Boolean Satisfiability Problem, `SAT`.

Definition 1.2.18. Let ϕ be a Boolean Formula. `SAT` $:= \{\langle\phi\rangle : \phi \text{ is satisfiable}\}$.

Theorem 1.2.19. *`SAT` is* $\mathbf{NP}-$*Complete*

Proof. We omit the proof as it is unnecessarily large for the purposes of this dissertation. An accessible version can be found in [28]. □

When we want to show completeness in this dissertation, we take as given a known complete problem and perform the required reduction. Below we give an example of a very straightforward reduction. This is taken from [28].

Definition 1.2.20. Let ϕ be a Boolean Formula in Conjunctive Normal Form such that all clauses have 3 literals. `3SAT` $:= \{\langle \phi \rangle : \phi \text{ is satisfiable}\}$.

Definition 1.2.21. Let $G = (V, E)$ be a graph. A clique is a subset of the set V in which every pair of vertices is connected. The `CLIQUE` asks if there exists a clique of size k.

Theorem 1.2.22. *`CLIQUE` is* **NP**$-$*Complete.*

Proof. To show completeness for `CLIQUE` we need to do two things; First, show membership in **NP** and then to show that an **NP**$-$Complete problem is reducible `CLIQUE`. The reduction is done from `3SAT` which we take for free that is **NP**$-$Complete. We will need to construct a function that transforms an instance of `CLIQUE` to `3SAT` and conversely.

i. We begin by showing membership in **NP**. To do this, we will use a clique of size k as a certificate. Then, we need to check if all pairs are connected, which can be done in polynomial time, hence `CLIQUE` $\in$ **NP**.

ii. The main idea of the reduction is that we convert a propositional formula ϕ into a graph G and conversely. Let ϕ be a CNF formula with 3 literals in each clause; That is, for example $\phi = (a \vee b \vee c) \wedge (x \vee y \vee z)$ and more generally: $\phi := \bigwedge_{i=1}^{k} (a_i \vee b_i \vee c_i)$. For ϕ to be true, at least one of each literals needs to be true, in each clause. Now, define G as follows: Partition V into k subsets, each of size 3. We call these subsets triplets. Each triplet corresponds to a clause and each vertex in the triplet corresponds to a literal. Arrange the edges as follows; Each edge connects vertices in distinct triplets and no edge connects the negation of another vertex; That is an $(a_i,\ a_i)$ is not allowed.

   (a) For the first case, suppose that there is a satisfying assignment to ϕ. Then we have a clique. Why? Because there is a satisfying assignment, in each clause there is at least one true literal. From each triplet, pick a true vertex. We obtain a size k clique.

   (b) Suppose that G has a clique. Then there is a satisfying assignment to ϕ. Why? By construction, the clique does not contain any two vertices from the sample triplet and hence each vertex in the clique comes from distinct triplets. Also, note that no contradictory vertices are connected, hence they cannot be in the clique. Assign to vertices in the clique a true value to the corresponding literals. Therefore, ϕ is satisfiable.

$\square$

Consider our standard computational model, the Turing Machine and its Non-Deterministic equivalent. Suppose we modify them by granting them some sort of "black box" that allows us to obtain certain information for free. Then, depending on what information we allow to be obtained, our models might be able to solve problems more easily. As an example, suppose that we grant a standard deterministic Turing Machine the ability to solve `SAT` in a single step, ignoring how the solution is actually obtained. Such modified machines are called Oracle Machines. More generally, we have as follows:

Definition 1.2.23. Let A be a language. An oracle for A is a device that can report whether given w, if $w \in A$. An oracle Turing Machine, M^A, is a standard Turing Machine but equipped with an A-oracle. The decision is $w \in A$? is made in a single computational step. We say that M computes relative to A.

Definition 1.2.24. We define $\mathbf{P}^A$ the set of languages decidable in polynomial time using a Turing Machine equipped with an A-oracle.

The class $\mathbf{NP}^A$ is defined analogously.

In concluding this part, we formally present (and recap) some of the main computational problems that will be used throughout the dissertation.

Definition 1.2.25. $\texttt{SAT} := \{\langle\phi\rangle : \phi \text{ is satisfiable}\}$

Definition 1.2.26. $\texttt{3SAT} := \{\langle\phi\rangle : \phi \text{ is satisfiable}\}$, ϕ is a Boolean Formula in Conjunctive Normal Form such that all clauses have 3 literals.

Definition 1.2.27. $\texttt{CLIQUE} := \{\langle G, k\rangle : G \text{ is undirected with clique of size } k\}$

Definition 1.2.28. $\texttt{VERTEX COVER} := \{\langle G, k\rangle : G \text{ is undirected with vertex cover of size } k\}$, where a vertex cover is a subset of the set of vertices such that they include at least one endpoint from each edge of G.

Definition 1.2.29. $\texttt{SET COVER} := \{\langle U, S, k\rangle : U$ is a universe set $, S$ is a collection of subsets of at most k elements that cover all elements in $, U\}$, where a set cover is a collection of subsets whose union is the universe.

Definition 1.2.30. $\texttt{DOMATIC NUMBER} := \{\langle G, k\rangle | G \text{ is a graph with domatic number } k\}$. For a set G, a dominating set is $D \subseteq V$ such that every vertex not in D is adjacent to some vertex in D. The domatic number is the maximum positive integer k such that V can be partitioned into k pairwise disjoint dominating sets.

Definition 1.2.31. `PARTITION` $:= \{\langle S\rangle |$
is a finite set of integers that can be partitioned into two subsets with equal sums$\}$

1.3 Computational Social Choice Theory

At this point, it makes sense to draw some connections between Social Choice Theory and Computational Complexity Theory - two seemingly unrelated fields. For a full account of the field, refer to [25], [1] or [8].

i. From Social Choice Theory to Computer Science: Social Choice Theoretic ideas and mechanisms can be applied in areas such as AI and multi-agent systems where selfish software agents have competing preferences. Another area is ranking and recommendation algorithms that select things according to certain criteria.

ii. From Computer Science to Social Choice Theory: In this direction, computer science techniques are applied to deal with Computational and Algorithmic aspects of Social Choice Theory.

In this dissertation, we focus on the latter - Computational and Algorithmic aspects of Social Choice Theory; First of all, we will consider the Winner Problem. An obvious property that we want our Voting Systems to possess is that winners can be determined efficiently. While many rules, among those used in practice, succeed in doing that, there exist computationally hard aggregation rules. In Chapter 2, we see how modifications of known $\mathbf{NP}-$Complete Problems bring the need for more refined complexity classes, specifically the class, $\mathbf{P}^{\mathbf{NP}_{||}}$ and place several pairwise comparison systems in this class. $\mathbf{P}^{\mathbf{NP}_{||}}$ is a strict superset of $\mathbf{NP}$, hence for these systems, winner determination is intractable.

Next, while it is impossible to design a strategy-proof preference-based Voting Rule, we will try to get around this result by using Computational Complexity to our advantage. In this case, computational complexity is a good thing and might provide us with some level of protection against strategic voting. In Chapter 3 we first see a simple algorithm that can manipulate many of our commonly used Voting Schemes. Then, we see how through a series of tweaks and modifications, manipulation becomes computationally intractable.

In passing and to connect the two fields, we present some basic polynomial time results for the winner problem. These results, although not complicated, cannot be found anywhere in the literature.

Theorem 1.3.1. *$\epsilon-$Winner $\in$ **P** if ϵ is a Scoring Protocol*

Proof. Construct a $n \times m$ array, where on the horizontal axis, we have some ordering of the m candidates and on the vertical axis, some ordering of the n voters. Then, entry (i, j) is given by either 1 or $m - k$, for k depending on the rule used. To pick the winner, we need to loop through the array and find the sum of each column - this gives us the scoring vector. From there, we need to pick the maximum element from the vector. Both can be done in polynomial time, in particular $O(n * m)$, hence the problem is in **P**. $\square$

Theorem 1.3.2. *$\epsilon-$Winner $\in$ **P** if ϵ is the Condorcet Protocol*

Proof. Represent the results of comparisons by a directed graph such that there is an edge from X to Y if and only if X is preferred to Y by a majority of voters. To determine the winner, it suffices to find the vertex with the most out-neighbours. It takes $O(n)$ steps to do this, hence the problem in **P**. $\square$

Theorem 1.3.3. *$\epsilon-$Winner $\in$ **P** if ϵ is the STV Protocol*

Proof. By the description of the Protocol, we know that it will run at most as many times as the number of candidates. This is $O(|A|)$, so the problem is in **P**. $\square$

Chapter 2

Parallel Access to NP and the Winner Problem

2.1 The Class $\mathbf{P}_{||}^{\mathbf{NP}}$

Consider the following standard **NP**−Complete problems; `VERTEX COVER, CLIQUE, DOMATIC NUMBER` and `TSP`, and observe what they have in common. They are all defined as Decision Problems of the form max $\geq m$ or min $\leq m$, for some $m \in \mathbb{Z}$. Now, suppose we modify these problems and ask more complicated questions - specific or exact facts about these maxima or minima [30]. For example, consider a variant of `VERTEX COVER` that asks specifically whether the minimum vertex cover $\tau(G)$ is odd. We call this problem the `MINIMUM ODD VERTEX COVER`. Or consider a variant of `DOMATIC NUMBER`, `EXACT DOMATIC NUMBER` which asks whether $\delta(G) = i$ for some $i \in \mathbb{Z}^{+}$. Taking `MINIMUM ODD VERTEX COVER`, it can be shown that the problem is **NP**-hard, however how we would prove that it is **NP**-Complete. That is, how to prove membership in **NP**? While it is easy to verify that $\tau(G) \leq m$; Simply, check that $V^{'} \leq k$ and then that every edge of G has at least one endvertex in $V^{'}$, it seems impossible to verify, in polynomial time, the parity of the minimum vertex cover [29]. To understand exact complexities of certain problems, natural extensions of **NP** have been developed. From Chapter 1, recall the class $\mathbf{P}^{\mathbf{NP}}$ which contains the languages that can be accepted in polynomial time by a Deterministic Turing Machine with access to an **NP** oracle. By restricting the oracle access and only allowing questions to be asked in parallel, we obtain the class $\mathbf{P}_{||}^{\mathbf{NP}}$.

Definition 2.1.1. The Class $\mathbf{P}_{||}^{\mathbf{NP}}$ contains exactly those languages that can by accepted in polynomial time by a Deterministic Turing Machine with the ability to make queries to an **NP** oracle, in parallel.

There are many different characterizations for this class and focusing purely on the intricacies and many results of/on $\mathbf{P}^{\mathbf{NP}}_{||}$ is outside the scope of this dissertation. The interested reader is referred to the following:

For our purposes, it suffices to understand how this class works and to keep in mind that standard complexity-theoretic notions such as membership, completeness and reductions, carry over in the regular sense. That is, a problem is $\mathbf{P}^{\mathbf{NP}}_{||}-$Complete, if it is in $\mathbf{P}^{\mathbf{NP}}_{||}$ and if its is $\mathbf{P}^{\mathbf{NP}}_{||}-$Hard.

Consider once again `MINIMUM ODD VERTEX COVER`. The following theorem, from [29], states Parallel Access completeness. The hardness proof can be found in [30].

Theorem 2.1.2. *`MINIMUM ODD VERTEX COVER` is $\mathbf{P}^{\mathbf{NP}}_{||}-$Complete.*

Proof. We omit hardness and show membership in $\mathbf{P}^{\mathbf{NP}}_{||}$. To show membership we simply need to find a polynomial time algorithm that will solve the problem with the aid of parallel-asked questions to some **NP** oracle. For the oracle, we take `MINIMUM ODD VERTEX COVER`. Then, we make the following list of questions: $< G, 1 >, < G, 2 >, ..., < G, n-1 >$. As the oracle returns the answers we accept if and only if $\tau(G)$ is odd. This can be done in polynomial time, hence `MINIMUM ODD VERTEX COVER` $\in \mathbf{P}^{\mathbf{NP}}_{||}$, as required. □

We will now introduce the $\mathbf{P}^{\mathbf{NP}}_{||}-$Complete Problem that will be required for the next section. We first show the **NP**−Complete Problem, `FEEDBACK ARC SET`. Then we define a version of `FEEDBACK ARC SET`, `FEEDBACK ARC SET MEMBER` that is complete with respect to parallel access to **NP**.

Definition 2.1.3 (`FEEDBACK ARC SET`)**.** Let G be a digraph. A feedback arc set of G is a set of edges that includes at least one edge from every cycle in G. Given $k \in \mathbb{Z}^+$, the `FEEDBACK ARC SET` problem asks whether there is a feedback arc set of size at most k.

NP-Completeness of `FEEDBACK ARC SET` can be found in [14].

Definition 2.1.4 (`FEEDBACK ARC SET MEMBER`)**.** Let G be an irreflexive and antisymmetric digraph and let $v \in V$. The modified problem `FEEDBACK ARC SET MEMBER` asks whether there is some minimum size feedback arc set that contains all edges entering v.

Theorem 2.1.5. *Feedback Arc Set Member is $\mathbf{P}^{\mathbf{NP}}_{||}-$Complete*

Proof. Omitted due to space. Instead, we give a rough idea. The full proof can be found in Chapter 5 of [29]. The proof that `FEEDBACK ARC SET` is **NP**-Complete was by a reduction from the `VERTEX COVER` problem. To show $\mathbf{P}_{||}^{\mathbf{NP}}$-Completeness of the `FEEDBACK ARC SET MEMBER` problem, the authors use a modified version of `VERTEX COVER` and adjust the reduction. □

2.2 $\mathbf{P}_{||}^{\mathbf{NP}}$ - Completeness of the Winner Problem under the Kemeny Rule

In this section, we will study the Winner Problem for the Kemeny Rule. We first describe the Kemeny Rule, first described by Kemeny in [20][1]. As with all voting protocols that we have seen so far, the Kemeny Rule is preferential; that is, it is designed for elections in which voters provide a full ranking of their preferences. Since this is the most complicated rule that we have seen so far, we first give an intuitive explanation of the rule. Initially, each voter provides a full ranking of their preferences with ties being allowed. The main aim of the Kemeny Rule is to provide some sort of compromise or, better, the most representative ranking. We want to end up with some ranking that best represents the collective preferences. To achieve this, we need to do two things: First, measure the agreement between rankings; For any ranking, measure how well it agrees with any other ranking. We then seek a ranking that will minimize the total disagreement among voters. From that ranking, we pick the winner.

Formally, the Kemeny Rule is described as follows:

Definition 2.2.1. The Kemeny Consensus is the collection of rankings closest to the rankings of the voters

Definition 2.2.2. A candidate wins the elections if he is the preferred candidate in a Kemeny Consensus

To deal with closeness and be able to minimize disagreement, we require some sort of distance function. This is the Kemeny Score, which we seek to minimize.

Definition 2.2.3. For any pair of rankings P, Q, define the distance, $dist(P, Q)$ as:

[1]More details on the workings of the Kemeny Rule can be found in [25], [18], [1] and [29].

$$dist(P,Q) := \sum_{\{x,y\}} d_{P,Q}(x,y)$$

where the pairs $\{x, y\}$ denote all candidates and the function $d_{P,Q}(x, y)$, assigns agreement scores as follows:

$$d_{P,Q}(x,y) := \begin{cases} 0 & \text{P and Q strictly agree} \\ 1 & \text{partially agree} \\ 2 & P\text{and Q strictly disagree} \end{cases}$$

The following functions are Kemeny Scores, which are used to define the computational decision problems.

Definition 2.2.4. Given a set of candidates C and a set of rankings,V,on C, the following are score functions:

i. Let P be a preference ranking, then:

$$Sc(C,P,V) := \sum_{Q\in V} dist(P,Q)$$

ii. Let $c \in C$. Then:

$$Sc(C,c,V) := \min\{Sc(C,P,V)|$$

P preference on C and c preferred candidate in P

$$\}$$

iii.

$$Sc(C,V) := \min\{Sc(C,P,V)|$$

P preference on C

$$\}$$

Given these functions representing the Kemeny Score, we are ready to define the computational problem of `KEMENY WINNER`. To do this we need to define a series of "helper" problems. These definitions are from [18] and [29].

Definition 2.2.5. i. `KEMENY SCORE`: Is $Sc(C, V) \leq k$, for $k \in \mathbf{Z}^+$?

ii. `CANDIDATE KEMENY SCORE`: Is $Sc(C, c, V) \leq k$, for $k \in \mathbf{Z}^+$?

iii. `KEMENY WINNER`: Is $Sc(C, c, V) \leq Sc(C, d, V)$, for all candidates d?

The main theorem, following the original work of [29] and [18], is the following:

Theorem 2.2.6. *`KEMENY WINNER` is $\mathbf{P}_{||}^{\mathbf{NP}}$−Complete.*

As with any Completeness proof, we need to show two things; membership and hardness. Unfortunately, the hardness proof is too complicated and big for the scope of this work. We will focus on the membership proof which highlights the workings of the class $\mathbf{P}_{||}^{\mathbf{NP}}$.

Lemma 2.2.7. *`KEMENY WINNER` $\in \mathbf{P}_{||}^{\mathbf{NP}}$.*

Proof. The first step involves finding an **NP**−Oracle. For this, we will use the `CANDIDATE KEMENY SCORE` which is clearly in **NP**. Given a candidate ranking c, we simply need to verify if its score is less than k, which can be clearly verified in polynomial time. The oracle will answer the `CANDIDATE KEMENY SCORE` question for each candidate.

Given a set of candidates C and a set of preference orderings V with respective cardinalities $|C|$ and $|V|$, observe that the score of each candidate is bounded above by $|V||C|^2$. This is by definition of the Kemeny Score function which essentially counts disagreements. Each voter contributes to a disagreement by potentially all voters in the worst case and the total contribution is the same we have $|V|$ for all voters times the total number of pairs which is simply $O(\binom{|C|}{2}) = |C|^2$. That is, the bound shows us the maximum possible disagreements.

We query in parallel for all candidates, which is by definition of the complexity class we are working in. That is, we simultaneously ask the oracle for the result of all possible values of k for all candidates. From there, the Kemeny Score is determined by identifying the smallest value of k for which the oracle answered yes.

Once all Candidate Kemeny Scores have been obtained we can easily identify the winner: $Sc(C, c, V) \leq Sc(C, d, V)$. This is found by simply comparing the scores and finding the minimum one - clearly done in polynomial time.

All these tasks can be done in Polynomial time, hence the problem is in $\mathbf{P}_{||}^{\mathbf{NP}}$, as required.

$\square$

Lemma 2.2.8. *FEEDBACK ARC SET MEMBER is polynomially reducible to KEMENY WINNER*

Proof. The full proof is omitted and can be found in [18] and [29]. □

Lemmas 2.2.7 and 2.2.8 complete the proof of 2.2.6.

At this point, it is worth noting the richness of the class $\mathbf{P}_{||}^{\mathbf{NP}}$, concerning Computational Social Choice Theory. The study of the Winner Problem began with the work of Bartholdi, Tovey and Tick in their 1989 paper on the Computational Hardness of determining the Winner under several Voting Schemes [4]. Indeed, they showed that the Kemeny Winner problem is $\mathbf{NP}-$Hard, however, they did not manage to close the gap and find the exact complexity of the problem [4], Since then, the intuition that winner determination for several of these voting mechanisms is (computationally) hard materialized through the class $\mathbf{P}_{||}^{\mathbf{NP}}$. Specifically, the following voting systems[2] have been classified ad $\mathbf{P}_{||}^{\mathbf{NP}}$-Complete:

i. The Dodgson Scheme; refer to [17]

ii. The Young Scheme; refer to [26]

iii. The Slater Rule; refer to [21] and [19]

[2]Note that these are voting schemes that have not been covered in this dissertation. The point here is to understand the importance of the class $\mathbf{P}_{||}^{\mathbf{NP}}$ rather than the exact details of each different scheme

Chapter 3

The Manipulation Problem

We encountered strategic voting and the Gibbard-Satterthwaite Theorem which roughly states that all voting schemes are manipulable [25]. Clearly, this is a bleak result, so we need to find a way to deal with this. Many ways have been proposed, [1], but in this dissertation, we consider Computational Complexity as a barrier to strategic voting. Specifically, while Gibbard and Satterthwaite state that every voting scheme is manipulable in practice, we consider the computational (in)tractability of a manipulative action; If, for a voting system ϵ, manipulation is intractable, then we consider ϵ as resistant. This is interesting in two ways; First of all, we can understand the shortcomings of our standard voting rules. Second, even if standard voting rules turn out to be easily manipulable, we can tweak them or design new ones. This is the approach we follow in this Chapter; In Section 3.1, we present a very simple Greedy Algorithm that solves the Manipulation problem in Polynomial Time for a variety of commonly used voting schemes. This is due to Bartholdi, Tovey and Trick, [6], who had the groundbreaking idea of using complexity as a shield against Manipulation [25]. Their seminal work has initiated a lot of studies in Manipulation and Computational Social Choice Theory. In Section 3.2 we show how through a series of tweaks and combinations, we can obtain 'new' protocols for which Manipulation becomes intractable. These results are due to Conitzer, Sandholm, Elkind and Lipmaa [10], [13]

3.1 Easy Manipulation via GreedyManipulation

We are given a set of voters N, a set of alternatives A, a voting rule ϵ, a manipulator $i \in N$ and a distinguished candidate. We are given the preference profiles of all $n \in N - \{i\}$ and the question is whether the manipulator can construct a preference profile that ensures p wins. If this can be answered in polynomial time, then the voting scheme in question is easily manipulable. A very simple algorithm, called `GreedyManipulation`, [6], solves the problem in polynomial time for voting schemes that satisfy Responsiveness and Monotonicity[1], in polynomial time and either outputs the required preference profile, whenever it exists, or concludes that this is not possible. The algorithm is surprisingly simple. We begin by placing p in the first place of the profile and then while there are unranked candidates, we place them in the next spot, without preventing p from winning. If at some point during the running of the algorithm, no such candidate can be placed, we return $\emptyset$, indicating that the required profile cannot exist. Formally, the algorithm is presented below in pseudocode.

Algorithm 2: `GreedyManipulation`

```
Data: $N$,$A$, $\epsilon$, $i$, $p$, $P_{-i}^N$
Result: $P_i$
initialization Rank $p$ first in $P_i$;
while $A - \{p\} \neq \emptyset$ do
    if $\exists\ b \in A - \{p\}$ that can be placed in the next spot in $P_i$ without preventing p from winning then
        Rank $b$ in the next spot;
    else
        $P_i \leftarrow \emptyset$;
        return $P_i$
    end
end
```

Theorem 3.1.1. *`GreedyManipulation` will construct P_i that makes p the winner, or conclude that such ordering does not exist, for any voting scheme ϵ that satisfies Monotonicity and Responsiveness. Furthermore, `GreedyManipulation` runs in Polynomial Time.*

Corollary 3.1.2. *If ϵ is one of Plurality, Borda, Maximin or Copeland, ϵ-Manipulation $\in \mathbf{P}$.*

The theorem and corollary stated above, are due to Bartholdi, Tovey and Trick, [6] and are the main results that we are going to prove in this section. Before we prove them, we provide some intuition on how the algorithm works by considering a simple example in the case of Borda Count.

[1]From Chapter 2

Example 3.1.3. Let $A = \{a, b, c, d\}$, $N = \{1, 2, 3\}$ and ϵ is the Borda Count. The preferences of voters $1, 2$ are given by $P_1 = P_2 = (a, p, b, c)$. Voter 3 is the manipulator and wants to construct a profile P_3 to ensure that p will win using `GreedyManipulation`. The first step is to rank p first. Now a has six points and p seven points, thus a cannot go in the second position of 3's ranking, since then a will end up with eight points. The solution is to put b, hence: $P_3 = (p, b)$. Again, a cannot go in the next places, since then a ties with p, so we put c next, with one point. Putting a in the last position ensures that p still wins, hence the output is (p, b, c, a) and the manipulator has his profile, as required.

We are now ready to present the proof of 3.1.1.

Proof. First, if `GreedyManipulation` outputs a preference ordering, then by construction, the ordering is correct and guarantees that p will win. We now need to prove that the algorithm will find the ordering, whenever it exists. For contradiction, suppose that the algorithm has terminated without having found the ordering but the ordering exists. Let $P^{'}$ be the preference profile that ensures p wins and suppose that the algorithm has terminated without having produced $P_i^{'}$ and let $U \subset A$ be the set of unranked candidates[2], upon termination of the algorithm. Call the partial ordering constructed by the algorithm P. We complete P as follows; Let u be the highest ranked element of U under $P^{'}$. Place u in the highest unassigned place of P. Given responsiveness and monotonicity we have as follows:

i. Since p wins under $P^{'}$, then by responsiveness, we have $\forall x \in A$, $s(P^{'}, x) \leq s(P^{'}, p)$. In particular: $s(P^{'}, u) \leq s(P^{'}, p)$.

ii. Under P, u is only ranked above all other candidates in U. Under $P^{'}$, u is ranked above all other candidates in U and possibly some other candidates as well. Hence, by monotonicity we have that $s(P, u) \leq s(P^{'}, u)$.

iii. Clearly, every candidate that is below p under P^i is also below p under P, hence by monotonicity we have that $s(P^i, p) \leq s(P, p)$.

We obtain the following chain of inequalities, $s(P, p) \geq s(P^{'}, p), \geq s(P^{'}, u) \geq s(P, u)$, which brings us to $s(P, p) \geq s(P, u)$. But this is a contradiction since upon running the algorithm in the first place, we terminated without being able to rank u. Hence, our initial assumption is wrong and `GreedyManipulation` will always find the correct profile.

To bound the running time of the algorithm, observe that the while loop will run at most $|A-$

[2]Note that by the initialization step of the algorithm, there will always be a partial ranking, even if that is the trivial ranking

$p|$, times, hence `GreedyManipulation` runs in Polynomial Time. Specifically, polynomial in the number of candidates. □

All the systems we have been considering in 2 are Monotonic, hence the corollary follows from there.

3.2 Hard Manipulation via Tweaking and Hybrid Protocols

In this section, we modify existing protocols to make manipulation a hard problem. The simplest reason for tweaking a protocol, instead of designing one from scratch is that by modifying existing protocols, desirable properties are preserved. The first approach involves tweaking by adding a preround. Under this very simple modification, we obtain a 'new' protocol that is hard to manipulate.

Definition 3.2.1. Let ϵ be a voting rule. We obtain ϵ+PREROUND as follows:

i. Pair the candidates; If there is an odd number, one candidate gets a bye

ii. For each pair, the candidate losing the pairwise election is eliminated. Candidates with a bye are never eliminated.

iii. On the remaining candidates, execute ϵ using the implicit votes.

In [10], Conitzer and Sandholm present a sufficient condition that makes manipulating protocols with prerounds **NP**-Hard. The theorem is rather complicated to state, but the main idea is that if we can reduce some `SAT` instance to a set of votes, satisfying some properties under ϵ, manipulation becomes hard. Their condition can be thought of as a reduction template and can be applied to many voting systems. The corollary to Conitzer's and Sandholm's condition is presented as a full theorem below.

Theorem 3.2.2. *Manipulation of ϵ+PREROUND is* **NP**−*Complete if* $\epsilon \in$ { *Plurality, Borda, Maximin* }

Proof. The proof of this is in Section 4 of [10] and relies on a reduction from `SAT`. The first part of that section is concerned with stating and proving the sufficient conditions. Next, it is applied to each of the voting systems mentioned in the theorem. For brevity and due to its high degree of complexity it is omitted. □

We will focus on a specific case of a tweaked protocol, Plurality with a pre-round and will show that the manipulation of that protocol is $\mathbf{NP} - \mathit{Complete}$. Following the definition of pre-round protocols, Plurality with a Preround is as expected; Initially, all candidates are paired up and for all pairs, whichever candidate loses the pairwise election is removed. Then, on the set of the remaining candidates, we run the standard Plurality rule. Following the simplified proof of Procacia in [2], we show $\mathbf{NP} - \textit{Completeness}$. As always, a completeness proof requires two things; Membership and Hardness. Membership is straightforward, as given a preference profile we can immediately, in polynomial time, check if the distinguished candidate p has been elected. The reduction part, using `SAT`, is more complicated and will consist of two parts; In the first part, we set up a couple of seemingly artificial sets and subsets. These are used to create an instance of the Plurality with a Pre-round which is then reduced to `SAT`. We focus on the reduction part as this is the interesting part; The details are below.

Corollary 3.2.2.1. *Manipulation of the Plurality with Pre-Round is* $\mathbf{NP} - \textit{Complete}$.

Proof. Let ϕ be a formula in Conjunctive Normal Form[3]. Let V be a set of variables, L a set of literals and C a set of clauses. Define D to be a set of dummy candidates, with $|D| = |C| + 1$ and A to be the set of candidates given by $A = \{p\} \cup L \cup U \cup D$. The instance of the Manipulation Problem is created as follows; Consider V and three subsets of V;

- The first subset consists of $4|C| + 2$ voters. For each, p is the top candidate and any dummy candidate is at the bottom.
- In the second subset, we have $4|C|$ voters for every $c \in C$ such that c is the top candidate and any dummy candidate is at the bottom.
- In the third subset, we have 4 voters for every $c \in C$, where each voter in their preference the literals that appear in c are ranked at the top, c is ranked after them and any dummy candidate is at the bottom.

Observe that the size of the first subset is simply $4|C| + 1$, the size of the second subset is $4|C|^2$ and of the third subset $4|C|$.

The construction of these subsets allows us in a sense to play with the number of voters (and thereby, the number of voters per candidate), given some boolean CNF formula ϕ and arrive at the desired result.

Now, all voters rank literals in L such that, for every variable, the variable and its negation are tied. During the pre-round, the pairings are done as follows; for every $v \in V$, v, v are

[3]A propositional formula in CNF is a conjunction of one or more clauses, where each clause is a disjunction of literals. More simply, a product of sums or an AND of OR.

paired together and every candidate in $\{p\} \cup C$ is paired with a candidate from D. Note that this pairing is possible since we have defined D to have size one more from C. During the pre-round, observe that all candidates from $\{p\} \cup C$ move to the next round, since any candidate from D is ranked at the bottom. Furthermore, since v is tied with v, the manipulator can choose which of the two proceeds to the next round. At this stage, we are ready for the reduction. We need to prove two directions;

i. Suppose that there exists some satisfying assignment to ϕ. We will show that there is a preference ordering that the manipulator can use that can cause the election of the distinguished candidate p. As mentioned, since v is tied with v, the manipulator can choose which of the two proceeds to the next round. If it is the case that all true values proceed to the next round, then p wins. Why? Candidate p has at least $4|C|+2$ from the first subset. Then, for each $c \in C$, at least one literal has proceeded from the pre-round (This is the manipulator's choice from the true literal). Thus, every voter from the third subset will have at least one literal ahead of c, leaving c with only the $4|C|$ points he received from the second subset. Therefore, p will win with $4|C| + 2$ votes.

ii. Now we need to show that if there exists a preference ordering that the manipulator can use that can cause the election of the distinguished candidate p, then there exists some satisfying assignment to ϕ. We take the contrapositive. Suppose that there is no satisfying assignment to ϕ; Then, p loses the election. Consider why; In the pre-round, by construction, all candidates from p C move to the next round and since v is tied with v, the manipulator can choose which of the two proceeds to the next round. However, the formula is unsatisfiable, meaning that there is no possible assignment of truth values to its variables that makes the entire formula evaluate to true. That is, at least one clause is false. Hence, there exists a clause $c^* \in C$ that did not proceed to the next round. We now count the number of votes that c^* receives and show that it is strictly more than the number of votes that p can receive. Candidate c^* receives $4|C|$ points from the voters in the second subset and an additional four points from the voters in the third group. Therefore, candidate c^* has at least $4|C| + 4$ points, but p can have at most $4|C| + 3$ points, hence p cannot win.

We have shown that this problem is equivalent to `SAT`, as required. □

An extension to making manipulation hard by tweaking existing voting systems is through Hybrid Voting Systems. Elkind and Lipmaa, [13], described a technique that involves combining two (or more) voting protocols to obtain a 'new' system; Under this modification, in most cases, manipulation becomes hard, even when the underlying mechanisms are easy to manipulate. Intuitively, a hybrid of two voting systems, for example, X and Y works first by running some rounds to eliminate some candidates using X and then running protocol Y on the remaining candidates. The following definition is from [13].

Definition 3.2.3. Let X and Y be voting protocols and consider the preferences of the voters. A Hybrid System, Hyb(X_k, Y), consists of two phases; In the first phase, the protocol executes k steps of X. Let S be the set of candidates, not eliminated in the first phase. In phase two, the protocol applies Y to S.

It turns out that many hybrid protocols are **NP**−Hard to manipulate. Elkind and Lipmaa prove hardness by imposing conditions that X, Y need to satisfy such that Hyb(X_k, Y) becomes hard to manipulate. We will consider protocols when Y is the Plurality Rule and a fairly abstract property on X, that is satisfied by the Borda and Maximin Rules, obtaining two hard-to-manipulate rules; Hyb(Borda, Plurality) and Hyb(Maximin, Plurality). These results are from [13].

Property 1

For any set $G = \{g_1, ..., g_N\}$, collection of subsets of G, $S = \{s_1, ..., s_M\}$ and $K \leq M$, there are $k^{'}, k^{'} \leq M$ and $T, T > 3N$, such that it possible to construct in polynomial time a set of $T + N(T-2) + 3N$ votes over the set of candidates $C^{'} \cup C^{''} \cup \{p\}$, where $C^{'} = \{c^{'}_1, .., c^{'}_N\}, C^{''} = \{c^{''}_1, .., c^{''}_M\}$, such that:

1. there are T voters that rank p first
2. for each $i = 1, ..., N$, there are $T - 2$ voters who rank $c^{'}_i$ first;
3. For each $i = 1, \ldots, N$, there are 3 voters who rank all $c^{''}_j$ such that $g_i \in s_j$ above $c^{'}_i$, and rank $c^{'}_i$ above all other candidates.
4. for any additional vote, when it is tallied with all other votes, the set of candidates eliminated in the first $k^{'}$ rounds is a subset of $C^{''}$ of size $M - K$
5. For any subset $S^{'}$ of S, of size $M - K$, one can design in polynomial time a vote that when tallied with other votes, guarantees that set of candidates eliminated in the first $k^{'}$ rounds is $\{c^{''}_i : s_i \in S^{'}\}$

Theorem 3.2.4. *A hybrid protocol of the form Hyb(X_k, Y), where Y is the Plurality Rule is* **NP**−*Hard to manipulate whenever protocol X satisfies Property* 1.

Before we proceed to the proof, at this point it is worth noticing the common theme that encompasses these results. We have devised certain sets (in the case of Conitzer [10]) and properties (in the case of Elikind [13]) that can be directly 'converted' into a voting scenario. The reduction then can be performed and hardness naturally follows. Similarly to the proof of 3.2.2.1, whereby we constructed some stylized and seemingly artificial sets, Property 1 is constructed in such a way that it can be directly used for a reduction to our main problem. The proof of 3.2.4 relies on a reduction from `SET COVER`, a well-known $\mathbf{NP}-\mathit{Complete}$ Problem. The `SET COVER` is defined as follows: Given some ground set G, a collection of subsets of G, S and an integer K, does there exist a $K-$cover of G?

Proof. The basic idea of the reduction is as follows; We construct a set of votes based on the sets G, S and on K and assume that they satisfy Property 1. Let $k = k^{'}$, and let p be the distinguished candidate. Now, the idea is to show that the manipulator will succeed, under the Hybrid Voting Scheme, if and only if they can find a set cover for G. We therefore need to prove the two implications. That is, we need to show that p wins if and only if the K candidates correspond to a set cover. We thus prove the two directions;

i. Suppose that the corresponding element is not covered; That is, all $c_j^{''}$ are eliminated. After the first run, we have that $c_i^{'}$ has $T+1$ votes while p has T votes, and thus cannot win.

ii. Now suppose that for $g_i \in G$ we have some $s_j \in S$ such that $g_i \in S$ but $c_j^{''}$ is not eliminated in the first round. At the start of the second phase each $c_i^{'} \in C$ has $T-2$ first-place votes but p has T votes. Thus p wins, as required.

□

Now we have shown that all voting systems X_k that satisfy Property 1 can yield a Hybrid Protocol with Y being the plurality rule are $\mathbf{NP}-\mathit{Hard}$ to manipulate. Now, Elkind and Lipmaa [13], showed that this is true when X is the Borda and Maximin Rules.

Corollary 3.2.4.2. *Property* 1 *is satisfied by the Borda and Maximin Rules. Hence Hyb(Borda, Plurality) and Hyb(Maximin, Plurality) are* $\mathbf{NP}-$*Hard to manipulate.*

An additional method is to construct the Hybrid of a Protocol with itself. Under this construction, the 'new protocol' is different from the original and apparently, computationally hard to manipulate. From 3.1, we know that the Borda rule can be manipulated easily, simply using `GreedyManipulation`. We consider the Hybrid of Borda with itself and using a reduction from `EXACT COVER BY 3-SETS` and show that manipulation of this Hybrid

Protocol is **NP**−Hard. The `EXACT COVER BY 3-SETS` problem is stated as follows; Given a ground set $G = \{g_1, ..., g_N\}$ with $N = 3L$ and S, a collection of 3−element subsets of G, does there exist an exact set cover of G? The main idea of the proof is similar to the theme so far. We construct a voting scenario that resembles the problem, thereby creating an instance of that problem, to which we are reducing and then showing equivalence to that problem.

Theorem 3.2.5. *A Hybrid Protocol of two instances of Borda Count is* **NP**−*Hard to manipulate.*

Proof. Begin by constructing two sets of voters; V', V'', such that $|V'| = 2N + 2$, $|V''| = (M + 1)(N + 1)$ and let $V = V' \cup V''$. For the candidates, let $C = C^g \cup C^s \cup \{c_0\} \cup \{p\}$, where $C = C^g = \{C^g_1, ..., C^g_N\}$, $C^s = \{C^s_1, ..., C^s_M\}$ and p is the distinguished candidate.

Now consider the rankings of voters from each of V', V'';

For $i = 1, ..., N - 1$ we have 2 voters from the set V' that rank the candidates as follows: $(c^g_{i+1}, c^g_{i+2}, ..., c^g_N, p, c^g_1, ..., c^g_{i-1}, C^s_i, c^g_i, C^s - C^s_i, c_0)$, where $C^s_N = \{c^s_j | g_i \in s_j\}$. 2 voters will have the following ranking: $(p, c^g_1, c^g_2, ..., c^g_{N-1}, C^s_N, c^g_N, C^s - C^s_N, c_0)$, where: $C^s_N = \{c^s | g_N \in s^j\}$. Then we have two more voters that will have the following rankings: $(c^g_1, c^g_2, ..., c^g_N, c_0, p, C^s)$ and $(c^g_1, c^g_2, ..., c^g_N, p, c_0, C^s)$.

Now, consider the set of voters from V''; For each of $i = 1, ..., N - 1$, we have $M + 1$ voters with rankings: $(c^g_{i+1}, c^g_{i+2}, ..., c^g_N, p, c^g_1, ..., c^g_i, c_0, C^s)$. Another $M + 1$ voters with rankings: $(c^g_1, c^g_2, ..., c^g_N, p, c_0, C^s)$ and another $M + 1$ voters with rankings: $(p, c^g_1, c^g_2, ..., c^g_N, c_0, C^s)$.

Now, set $k = M - \frac{N}{3}$ and set preferences such that everyone in C^s has the same Borda score. The manipulator's vote will determine which k are eliminated.

We now show equivalence of the above to `EXACT COVER BY 3-SETS`.

i. Suppose that the set of candidates from C^s that survived the first round forms a set cover. By construction of the sets we have that for each candidate c^g_i and any $j = 1, ..., N$, there are two voters in V' who rank him in the jth position and two voters in V'' who rank him in the $(N + 2)$nd position. The Borda score of p will therefore be higher, hence p wins. Consider why; The Borda score of any candidate from C^g is

$$\sum_{t=m-k-N}^{m-k-1} 2t + 2(m - k - N - 2)$$

while the Borda score of p is:

$$\sum_{t=m-k-N}^{m-k-1} 2t + 2(m-k-N-2) + (m-k-N-2)$$

ii. For the other direction, suppose that the set of candidates from C^s that survived the first round does not form a set cover. Pick $g_i \in G$ such that g_i is not covered. By construction, the voters in V' will prefer c_i^g to all candidates in $C \cup \{c_0\}$, meaning that p's score will be lower and hence cannot win.

Therefore, we have shown that under the given construction this problem reduces to `EXACT COVER BY 3-SETS`, hence the problem is **NP**−Hard, as required.

□

3.3 Related Problems and Discussion

3.3.1 Related Problems

We consider the simplest case of the Manipulation Problem - The Single Manipulator case. The reason for this was given at the beginning of this section; We considered a very simple greedy algorithm that can be used to manipulate several well-known voting schemes and then we saw how through very simple tweaks the same schemes become hard to manipulate. While this gave us a very structured and unified theme, there are many avenues that can be taken when considering the manipulation problem. In closing this section, we provide some variations and modifications of the manipulation problem and provide some useful references for the interested reader. The simplest modification is to have a coalition of manipulators; In this case, we have a subset of the voters that jointly aim to strategically vote such that some distinguished candidate p wins the election. This problem was introduced by Conitzer, Lang and Sandholm in [9] and [11]. The same authors introduced weights to the (single and coalitional) manipulation problem, meaning that some voters are more important than others. Another modification is to distinguish Constructive versus Destructive Manipulation. In the former case, as we have been seeing so far, voters aim to make their candidate win. Analogously, voters might seek to prevent some candidate from winning, without actually caring about the winner. Recall that in 1, this was our first contact with the concept of strategic voting. An interesting problem is to understand the size of coalitions that turn the coalitional manipulation from an easy to a hard problem or the size of candidates that turn the problem from easy to hard. The following two theorems interestingly illustrate this for the Copeland Rule and are due to [11] and [9].

Theorem 3.3.1. *For elections with at most three candidates, (constructive) manipulation of the Copeland Rule is in* **P**.

The proof of this considers the case of exactly three candidates and the authors construct a polynomial time algorithm for manipulating the Copeland system.

On the contrary, if we have at least four candidates in the election, the same voting rule becomes hard to manipulate as the next theorem shows.

Theorem 3.3.2. *For elections with at least four candidates, (constructive) manipulation of the Copeland Rule is* **NP**−*Complete.*

Proof. The full proof is done by a using reduction from `PARTITION`. □

Bibliography

[1] *Handbook of Computational Social Choice.* Cambridge University Press, 2016. 6, 17, 21, 25

[2] Roy Schwartz Yoav Wilf Ariel D. Procaccia, Amit Goldstein. Mathematical foundations of ai lecture 7. 2008. 29

[3] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach.* Cambridge University Press, 2009. 9

[4] J. Bartholdi, C. A. Tovey, and M. A. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165, 1989. 24

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

[6] J. J. Bartholdi, C. A. Tovey, and M. A. Trick. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6(3):227–241, 1989. 25, 26

[7] Jean-Pierre Benoıt. The gibbard–satterthwaite theorem: a simple proof. *Economics Letters*, 69(3):319–322, 2000. 9

[8] Yann Chevaleyre, Ulle Endriss, Jérôme Lang, and Nicolas Maudet. A short introduction to computational social choice. In Jan van Leeuwen, Giuseppe F. Italiano, Wiebe van der Hoek, Christoph Meinel, Harald Sack, and František Plášil, editors, *SOFSEM 2007: Theory and Practice of Computer Science*, pages 51–69, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. 17

[9] Vincent Conitzer, Jerome Lang, and Tuomas Sandholm. How many candidates are needed to make elections hard to manipulate?, 2003. 34, 35

[10] Vincent Conitzer and Tuomas Sandholm. Universal voting protocol tweaks to make manipulation hard. In *Proceedings of the 18th International Joint Conference on Ar-*

tificial Intelligence, IJCAI'03, page 781–788, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc. 25, 28, 32

[11] Vincent Conitzer, Tuomas Sandholm, and Jérôme Lang. When are elections with few candidates hard to manipulate? *J. ACM*, 54(3):14–es, jun 2007. 34, 35

[12] Walter Dean. Computational Complexity Theory. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2021 edition, 2021. 9

[13] Edith Elkind and Helger Lipmaa. Hybrid voting protocols and hardness of manipulation. In Xiaotie Deng and Ding-Zhu Du, editors, *Algorithms and Computation*, pages 206–215, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. 25, 30, 31, 32

[14] Michael R Gary and David S Johnson. Computers and intractability: A guide to the theory of np-completeness, 1979. 20

[15] Allan Gibbard. Manipulation of voting schemes: A general result. *Econometrica*, 41(4):587–601, 1973. 9

[16] Oded Goldreich. *P, NP, and NP-Completeness: The basics of computational complexity*. Cambridge University Press, 2010. 9

[17] Edith Hemaspaandra, Lane A. Hemaspaandra, and Joerg Rothe. Exact analysis of dodgson elections: Lewis carroll's 1876 voting system is complete for parallel access to np, 1999. 24

[18] Edith Hemaspaandra, Holger Spakowski, and Jörg Vogel. The complexity of kemeny elections. *Theoretical Computer Science*, 349(3):382–391, 2005. 21, 22, 23, 24

[19] Olivier Hudry. On the complexity of slater's problems. *European Journal of Operational Research*, 203(1):216–221, 2010. 24

[20] John KEMENY. Mathematics without numbers. *Daedalus.*, 88(4), 1959. 21

[21] Michael Lampis. Determining a slater winner is complete for parallel access to np, 2021. 24

[22] Christos H Papadimitriou. Computational complexity. In *Encyclopedia of computer science*, pages 260–265. 2003. 9, 12

[23] Pavel Pudlák. *Logical foundations of mathematics and computational complexity: A gentle introduction*. Springer, 2013.

[24] Jörg Rothe. *Complexity Theory and Cryptology. An Introduction to Cryptocomplexity.* 01 2005.

[25] Jörg Rothe. *Economics and Computation: An Introduction to Algorithmic Game Theory, Computational Social Choice, and Fair Division.* Springer Publishing Company, Incorporated, 1st edition, 2015. 6, 17, 21, 25

[26] Jörg Rothe, Holger Spakowski, and Jörg Vogel. Exact complexity of the winner problem for young elections, 2001. 24

[27] Mark Allen Satterthwaite. Strategy-proofness and arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10(2):187–217, 1975. 9

[28] Michael Sipser. *Introduction to the Theory of Computation.* Course Technology, Boston, MA, third edition, 2013. 9, 10, 14

[29] Holger Spakowski. Completeness for parallel access to np and counting class separations. In *Ausgezeichnete Informatikdissertationen*, 2005. 19, 20, 21, 22, 23, 24

[30] Klaus W. Wagner. More complicated questions about maxima and minima, and some closures of np. *Theoretical Computer Science*, 51(1):53–80, 1987. 19, 20

[31] Avi Wigderson. *Mathematics and computation: A theory revolutionizing technology and science.* Princeton University Press, 2019. 9