2009

Improving Dodgson scoring techniques

Jason A. Covey

Improving Dodgson Scoring Techniques

by

Jason A. Covey

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Supervised by

Dr. Christopher M. Homan
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York
April 6 2009

Approved By:

Dr. Christopher M. Homan
Assistant Professor
Chair

Reader
Dr. Edith Hemaspaandra
Professor

Observer
Dr. Stanisław P. Radziszowski
Professor

Thesis Release Permission Form

Rochester Institute of Technology

Golisano College of Computing and Information Sciences

Title: Improving Dodgson Scoring Techniques

I, Jason A. Covey, hereby grant permission to the Wallace Memorial Library to reporduce my thesis in whole or part.

Jason A. Covey

Date

Dedication

To my grandmother Marjorie.

Acknowledgments

First and foremost I would like to thank my adviser, Dr. Christopher M. Homan, who first introduced me to, and got me hooked on, the field of theoretical computer science. I also appreciate the time that Chris took to teach me better scientific writing. He has been a great source of wisdom and inspiration over my many years at RIT.

I extend my gratitude to the other members of my committee, Dr. Edith Hemaspaandra and Dr. Stanisław Radziszowski. The many courses that I attended with them were both interesting and invaluable to my education.

Other professors I would like to thank at RIT for their contributions to my character and education are, from the Computer Science Department, Prof. Warren Carithers, Dr. Zack Butler, Dr. Hans-Peter Bischof, Dr. Minseok Kwon, Dr. Leon Reznik, Dr. Sidney Marshall, Dr. Axel Schreiner (I believe I can do anything if I can do his homework assignments), Prof. Trudy Howles, and Prof. William Childs. From the Music Department, I would like to thank Dr. Jonathan Kruger, from the Mathematics Department, Dr. Matthew Coppenbarger, and from the Philosophy Department, Dr. Katie Terezakis, Dr. Evan Selinger, and Dr. Brian Schroeder.

Next I thank my co-authors in [CCF+09], the conference paper in which the new algorithm presented here is a part of, for their contributions to the Dodgson election problem and insights into my own work.

Finally, I would like to thank Rev. Dr. E. Larraine Frampton, pastor of the Lutheran ministries at RIT, for her friendship and conversations which helped me through many troubles at RIT.

Abstract

The Dodgson score problem is part of the Dodgson election scheme invented by Charles Dodgson and presented in [Dod76]. One of the system's strengths (and motivations for its study) is that it satisfies the Condorcet criterion (which states that any candidate who defeats all other candidates in pairwise elections will be declared the winner). It is unfortunate, though, that in a given election no Condorcet winner may exist. Dodgson's election system chooses the winner "closest" to being the Condorcet winner in the sense that it requires the shortest sequence of edits (swapping of adjacent candidates in the voters' preference rankings) to the votes in order to make it one. The length of this sequence is known as the Dodgson score.

The problem of finding the Dodgson score of a candidate is computationally intractable. Thus an approximation is necessary. This paper puts forth `MCDodgsonScore`, a polynomial-time computable $(\ln(m) + 1)$-approximation of that problem. This approximation is optimal, meaning that an approximation with an asymptotically tighter error bound does not exist, as shown in [CCF+09].

`MCDodgsonScore` builds on a technique introduced by Homan and Hemaspaandra in [HH06]. A nice feature of `MCDodgsonScore` is that, when treated as its own voting rule, it will also satisfy the Condorcet criterion.

Contents

Chapter 1

Introduction

1.1 Voting Theory and Dodgson Elections

Starting with a group of voters who have ranked a group of candidates from most-preferred to least-preferred, a voting mechanism (also called voting system, or voting rule), is a method of taking those rankings and mapping them into one ranking (which may contain ties) for the entire group. There are many haphazard ways to do this, but researchers constantly seek out the voting mechanisms that provide the best representation of the group's preferences.

Of particular importance is the problem of finding the candidate that is most preferred, i.e., the winner. A natural way, proposed by Marie Jean Antoine Nicolas de Caritat, marquis de Condorcet in [Con85], to choose the winner is to pick the one that defeats all others in pairwise elections. Winners who satisfy this condition are known as Condorcet winners and elections that guarantee this behavior are said to satisfy the Condorcet criterion. It is unfortunate that, due to reasons left to Chapter 2, a voting method cannot be constructed based on the criterion alone.

To address this, researchers tried to find a way of choosing the candidate closest to being a Condorcet winner. Two methods for doing this involve making a series of edits in the votes in order to attain a candidate that is a Condorcet winner. The candidate with the shortest possible sequence of edits is declared the winner. The first method, proposed by Young in [You77], deletes votes from the election altogether. A second voting rule, given

in [Dod76] by Charles Dodgson, defines an edit as a swapping of adjacent candidates in a single vote. The length of the shortest series of swaps needed to make the candidate a Condorcet winner is the candidate's Dodgson score.

Unfortunately, the Dodgson election problem is computationally intractable. Bartholdi, Tovey, and Trick, in 1989, found that determining the solution of the decision problem version of determining the Dodgson score of a candidate is NP-complete, and that solving the decision problem version of the winner problem was NP-hard [BTT89b]. Hemaspaandra et al. improved upon their results, and showed that the winner decision problem is Θ_2^P-complete, i.e., complete for the class of problems solvable in polynomial time with the allowance for no more than $O(\log(n))$ queries to an NP oracle [HHR97].

Given these complexity results, implementing an exact solution is out of the question, so researchers began searching for heuristics. Two results leveraged the fact that in most political settings the number of voters greatly outnumbers the size of the candidate set. One of these results, put forth by McCabe-Dansted, Pritchard, and Slinko, uses Tideman's rule to find heuristically the winner [MDPS06].

In the second result, Homan and Hemaspaandra [HH06] use a heuristic similar to McCabe-Dansted et al. [MDPS06], but has the added property of being self-knowingly correct: when the algorithm outputs "definitely" as part of its result, then the score it provides with it is definitely the Dodgson score.

The `MCDodgsonScore` algorithm presented in this thesis builds upon the work in Homan and Hemaspaandra by actually modifying the votes, rather than just passing through them, but not enough to destroy the polynomial runtime. It also preserves the property of self-knowing correctness.

1.2 Applications of Voting Theory

Voting theory has several applications in practice. The first and most obvious area is in politics. The most widely used election mechanism (either as-is or modified) in politics is

called plurality. This voting rule elects the winner who receives the most first-place votes. This method of voting is sometimes called "tops-only" (as opposed to preferential) since it does not consider the position of any other candidate beyond the highest in a voter's preference ranking.

More complicated methods can be used in areas, such as artificial intelligence, for, e.g., the aggregation of web pages [DKNS01], avoiding spam results in web searches [FKS03], collaborative filtering [PHG00], and planning in automated multi-agent systems [ER91, ER93].

Other applications can be found in heating and cooling systems. For instance, The Golisano College of Computing and Information Science at Rochester Institute of Technology has one boiler per floor. However, each room may read a different temperature and it is wasteful for the furnace to stay on when it is not needed, just in case one room wants heat. Thus each room has its own temperature sensor and when that sensor goes too high or low from the norm it votes as to whether to turn the furnace on or off. The preferences of the rooms are polled periodically and the furnace reacts according to the aggregated preference of the group.

1.3 Organization of the Thesis

Chapter 2 of the thesis, Social Choice Preliminaries, will give the reader a mathematical introduction to election theory and some of the properties an election system can have and criteria it can satisfy.

Chapter 3 will go more deeply into the study of Dodgson elections, discussing the algorithmic and mathematical tools used to study them, and walking through a famous reduction involving Dodgson election related problems. This chapter will also discuss the `GreedyWinner` algorithm developed in [HH06] (since a proposed improvement will be detailed in Chapter 4). The `GreedyWinner` algorithm is a heuristic to the question: given a candidate and an election, is this candidate the winner?

Chapter 4 will give an algorithm that extends the `GreedyScore` algorithm as well as prove theorems associated with it, such as the polynomial runtime and the approximation bound. Chapter 4 will also show that when `GreedyScore` outputs a definitely correct answer, so does our algorithm.

Chapter 5 will conclude the thesis as well as present open questions and suggestions for further research.

Chapter 2

Social Choice Preliminaries

This chapter will give the reader an overview of the mathematical and computational theory behind Dodgson elections. First we will define the formal model of an election and then discuss specific voting mechanisms. Later, we will define some terms and notions from computational complexity theory that are useful in studying elections.

2.1 Formal Election Model

An election system has two main inputs: a set C containing m candidates, and a collection (multiset) V holding n votes. Note that V is not a set since it can contain more than one copy of the same vote (i.e., two or more people vote the same way). Since V is a multiset it is helpful to define some notation over multisets rather than sets. Given a vote $v \in V$, the collection $V - \{v\}$ means the collection V minus exactly one instance of the vote v. The collection $V \cup \{v\}$ means the collection V with exactly one additional instance of the vote v added into it.

A vote is a total ordering of the candidates. Let $C = \{c_1, c_2, c_3, \cdots, c_n\}$. Then a valid vote is denoted $(c_{b_1} \succ c_{b_2} \succ c_{b_3} \succ \cdots \succ c_{b_n})$ where each b_i is distinct and $1 \leq b_i \leq n$. We use the symbol $c \succ d$ to denote that candidate c is immediately preferred candidate d in the ranking (they are adjacent). We use the symbol $c >_v d$ to denote that candidate c is preferred over candidate d in vote v, but c may or may not be adjacent to d. The symbol $c \succ_v d$ denotes that candidate c is preferred over candidate d in vote v, and c is adjacent to

d in the vote.

Let $\mathcal{L}(C)$ be the set of all possible votes for candidates in C, then a preference profile may be seen as an n-tuple in $\mathcal{L}(C)^n$. More simply put a preference profile is a collection of the voters' rankings of the candidates. For a given preference profile $\langle \succ_1, \ldots, \succ_n \rangle \in \mathcal{L}(C)^n$, $i \in V$, and $c \in C$, let $c(>_i)$ denote $||\{d \in C \mid c >_i d\}||$, or the number of candidates which c defeats in the i^{th} vote. This is the same as the position of the candidate in the vote.

An election system is a preference profile together with a voting rule (also called voting method or voting mechanism). When we input the preference profile to an algorithm which computes a voting rule, the algorithm's output is a ranking of the candidates from most preferred to least preferred, and there may be ties between them. The first-place candidates are known as the winners.

2.2 Election System Criteria

This section presents criteria that one can use to evaluate an election system. The criteria presented here are the Condorcet criertion and the criteria studied by Arrow.

2.2.1 The Condorcet Criterion

The Condorcet criterion examines the relationship between one candidate and the others when compared in pairwise elections. A pairwise election is one that involves only two candidates. We consider the positions of each of the two candidates in each vote. The candidate who is ranked higher in the most number of the votes is the winner of the pairwise election.

This criterion is simple and states that a candidate who, when compared pairwise against each other candidate, defeats them all by a strict majority, should be declared the winner [Con85]. Any voting method which holds true to this principle is said to be a Condorcet method. This criterion seems like a logical one that every voting method should

satisfy. But not every voting method, even those widely used today, does so. It is a known folklore result that the plurality voting rule is not a Condorcet method and it is quite simple to show this with a counterexample.

Theorem 1. *Plurality is not a Condorcet method*

Proof. Plurality is a simple voting method in which the candidate ranked first by the most voters wins. Its simplicity makes it widely used in politics. Consider an election with 50 voters where $C = \{\text{Bush}, \text{Kerry}, \text{Nader}\}$ and $V = \{20 \text{ of } (\text{Bush} \succ \text{Kerry} \succ \text{Nader}), 15 \text{ of } (\text{Kerry} \succ \text{Bush} \succ \text{Nader}), \text{and } 15 \text{ of } (\text{Nader} \succ \text{Kerry} \succ \text{Bush})\}$. The plurality mechanism assigns a score of 20 to Bush, 15 to Kerry, and 15 to Nader. Bush is therefore declared the winner.

If we compare each candidate pairwise we will see that Kerry $\succ$ Bush 30-to-20 and Kerry $\succ$ Nader 35-to-15. Kerry is clearly the Condorcet winner. Since the plurality mechanism did not output Kerry as the winner, it fails to be a Condorcet method. □

Using the Condorcet criterion alone to construct a voting mechanism sounds tempting. It would be easy to just compare each pair of candidates and find the one that pairwise defeats all the others. However, there may not always be a Condorcet winner in the election and therefore we cannot use it as an election method.

For instance, consider an election where $a, b, c \in C$ and $V = \{(a \succ b \succ c), (b \succ c \succ a), (c \succ a \succ b)\}$. There is no Condorcet winner. It can easily be seen that each candidate is beaten by one of the other two. Therefore no candidate can be declared a winner! This condition is known as Condorcet's paradox.

2.2.2 Arrow's Impossibility Theorem

Arrow's impossibility theorem [Arr63] puts forth five axioms and states that no election system can satisfy all five of them at the same time.

1. Non-dictatorship: The voting rule cannot simply output the ranking of a single voter and must take the preferences of multiple voters into consideration.

2. Unrestricted domain: The voting rule must consider the preferences of all voters in the election when determining the output ranking and it must produce the same ranking every time the election is run on the same votes (no randomness is allowed).

3. Monotonicity: If a candidate has a specific position in the output ranking, then promoting that candidate in any individual vote or votes should not make that candidate have a lower position in the output ranking.

4. Non-imposition: The voting rule is a surjection, meaning that for every possible output ranking there is some input preference profile which results in it.

5. Independence of irrelevant alternatives (IIA): Given the output ranking of an election, if we introduce a new candidate to the election the rankings of the old candidates, with respect to each other, should remain the same.

Arrow's theorem holds only if the voters cast their votes independently, without knowledge of each other's votes. If voters actually collaborate to make sure there are no cyclical majorities (i.e., the votes are no longer independent), then a voting system can be constructed which satisfies all of Arrow's axioms [CT65]. In particular people have considered restricting the preference profiles that are admissible as input to a voting rule. In these cases we lose the unrestricted domain criterion, but can gain another criterion in the process. For instance, the Dodgson election system does not satisfy the IIA criterion, but we can consider only preference profiles put through the mechanism which would exhibit the behavior of IIA.

Definition 2. *Given an election system E, let E_{IIA} be the set of preference profiles of E where the independence of irrelevant alternatives criterion is satisfied. A specific instance of E_{IIA} is created with n candidates and has a total order of the candidates as an output. If one or more of those candidates is removed and the election run again then the new output of the election should be identical to the first output less the removed candidate. In other words, removing a single candidate, or arbitrary numbers of candidates for that matter, does not affect the outcome of the election with respect to the remaining candidates.*

2.3 Election Systems

Although the focus of this paper is on Dodgson elections, it is worthy to note that there are many distinct election systems studied by researchers. They vary widely in terms of rules and mechanisms, and complexity and fairness. This section defines both Dodgson elections and some other widely studied systems.

As stated earlier, there are many opinions as to what constitutes a fair election. Therefore, many rules have turned up. Among the simplest of these rules is the plurality election. There are also rules defined in terms of other rules, such as the hybrid elections studied in [EL06]. For each system considered, we will show how it works on the following example: three candidates, $C = \{a, b, c\}$, and fifty votes, $V = \{20 \text{ of } (a \succ b \succ c), 15 \text{ of } (b \succ a \succ c), 15 \text{ of } (c \succ b \succ a)\}$.

2.3.1 Dodgson Elections

The Dodgson election system is based on the Condorcet criterion and tries to find the candidate closest to being the Condorcet winner. For every candidate, the method computes the Dodgson score of that candidate. The candidate(s) with the smallest Dodgson score are declared the winner(s). The Dodgson score of a candidate is the shortest sequence of edits that need to be made to the collection of votes in order for that candidate to become the Condorcet winner. Finally, an edit is one swap of two adjacent candidates in the preference ranking of one vote.

In particular, for any vote v and any c, $d \in C$, if $c \succ_v d$, let $Swap_{c,d}(v)$ denote the vote v', where v' is the same total ordering of C as v except that now $d \succ_{v'} c$. If c is not adjacent to d then $Swap_{c,d}(v)$ is undefined. In effect, a swap causes c and d to switch places, but only if c and d are adjacent.

In our example, the Dodgson score of b is 0. This is because b defeats both a (30-to-20) and c (35-to-15) in pairwise elections. Since b defeats all other candidates in a pairwise election that makes b the Condorcet winner. Condorcet winners always have a Dodgson

score of 0 and will be output as one of the winners in this mechanism. This makes the system a Condorcet method.

The Dodgson score of a is 6. Candidate a already defeats c and therefore needs no improvement against it; however, it loses to b by 10 votes. We have swaps between a and b that will improve b's score available to us in votes of the form $(b \succ a \succ c)$. If we change six of these votes, a will defeat b by one vote, making it the Condorcet winner.

The Dodgson score of c is 22. Candidate c loses to both b and a by 20 votes each but we can follow roughly the same procedure we used to calculate the score of a. We need to make 11 swaps between b and c and 11 swaps between a and c for c to become the Condorcet winner. These swaps are available in votes of the form $(b \succ a \succ c)$ and $(c \succ b \succ a)$.

2.3.2 Young Elections

The Young Election [You77] is similar to Dodgson elections in that it also elects the candidate closest to the Condorcet winner. The definition of "closest" in a Young election is different, though. In this system if a vote hurts a candidate, then the vote is dropped from the election completely. The candidate who needs the least number of votes dropped to be declared the Condorcet winner is the winner of the Young election.

The Young score of b is 0, since b defeats a and c already. The Young election is, like the Dodgson election, a Condorcet method.

Candidate a has a Young score of 6. If we drop six votes of the form $(c \succ b \succ a)$ then a will defeat b by one vote. Dropping of these particular votes has no effect on the pairwise election of a and c.

The Young score of c is 11. 11 votes of the form $(a \succ b \succ c)$ should be dropped and this will simultaneously cause c to gain against both b and a.

2.3.3 Single Transferable Vote

Single transferable vote [BO91] is a widely used voting rule (elections in Ireland, the Australian senate, and local elections in Scotland, to name a few, use it). The goal of the rule is to minimize the number of votes that are wasted, by transferring them from weaker candidates to stronger ones. The rule takes a quota and a number of desired winners (i.e., available seats) as part of its input. It may be possible for the rule to produce more winners than seats depending on the quota used, therefore care must be taken to avoid to avoid this possibility when choosing the quota.

Single transferable vote must have a set quota, q, to function properly. The quota q is most commonly set to $(\frac{\text{\# of votes}}{\text{\# of seats}+1}) + 1$. This is called the Droop quota. The voting rule works in several steps that repeat as many times as needed until all seats are filled. Any candidate who has been ranked first in a number of votes greater than or equal to q is declared a winner. Second, we check to see if the required number of seats had been filled, if yes, then the algorithm terminates, if no, then we check to see if there have been any previous winners declared. If previous winners have been declared, and the number of votes where they are ranked first is greater than q, then those votes are given to the next-choice candidates, i.e., each vote for a winning candidate is transferred to other candidates according to the next preference in the voter's ranking. If no one still has more than q first-place, then the candidate with the fewest votes is eliminated, and his votes are transferred to the next-choice candidates. The process repeats until all seats are filled.

In our example, the Droop quota would be $(\frac{\text{\# of votes}}{\text{\# of seats}+1}) + 1 = (\frac{50}{1+1}) + 1 = 26$. No candidate has 26 votes, so we must eliminate the candidates with the fewest first=placed votes. In this case it is both b and c. This leaves only a as the declared winner with b and c tied for last. Single Transferrable Vote is not a Condorcet method and our example provides a proof of this by counter-example.

If we had two seats to fill instead of just one, we can change the quota to be $(\frac{\text{\# of votes}}{\text{\# of seats}+1})+1 = (\frac{50}{2+1}) + 1 = 18$ (rounding up to the next whole vote). Candidate a surpasses this quota with 20 votes and is immediately elected. The other two candidates have 15 votes each.

Since a only needed 18 to win, the remaining two votes are transferred to b, who is the second preferred candidate in these surplus votes. This gives b a score of 17 and c a score of 15. Since neither b or c meet the quota, the candidate with the lowest score, c, is dropped. This leaves only b to get the second seat.

2.3.4 Borda Count

The Borda Count [dB81] is a simple election system that computes a score for each candidate and the candidate with the greatest score wins. Given m candidates, we go through the votes and for each vote the candidate adds $m - 1$ to his score if he is the voter's top choice, $m - 2$ if he is the voter's second choice, and so on, so that he adds nothing to his score if he is last.

It is named for Jean-Charles de Borda, who created the system in 1770. The Borda system is currently being used by the National Assembly of Slovenia, among other organizations and countries.

Using our example, votes of the form $(a \succ b \succ c)$, c would get 0 points, b would get 1 point, and a would get 2. In votes of the form $(b \succ a \succ c)$, c gets 0 points, a gets 1 point, and b gets 2 points. In votes of the form $(c \succ b \succ a)$, c gets 2 points, b gets 1 point, and a gets 0 points. This gives c a total of 30 points, b a total of 65, and a a total of 55. The system would declare b the winner, followed by a in second place, and c in last.

2.3.5 Maximin

Maximin (see [EL06]), like Borda, computes a score for each candidate. A candidate's score in a pairwise election is the number of voters that prefer it over the opponent. A candidates number of points is the lowest score it gets in any pairwise election. The candidate with the highest score wins.

In our example, candidate a defeats c by 20, candidate b defeats a by 10, and c by 20. Candidate c defeats noone. Thus the scores of a, b, and c, are, respectively, 20, 10 and 0,

therefore a is elected the winner. Our examples provides a proof by counterexample that Maximin is not a Condorcet method since b was not elected as winner.

2.3.6 Binary Cup

The winner of the Binary Cup (see [EL06]) is determined by a tournament. The election consists of $\lceil \log m \rceil$ rounds where each candidate is paired up against another. The way candidates are seeded against each other in the tournament is usually an input to the algorithm that will determine the winner. If there are an odd number of candidates, one of them gets a bye to the next round. The winning candidates in each round then advance to the next. The actual tournament schedule can be given to the system as a parameter.

For our example, in the first round, we let candidate a go against c and give b a bye to the next round since it is in the lead. Candidate a defeats c in a pairwise election and therefore moves on to go against b. b defeats a and is therefore declared the winner. If we modified the tournament to give c the bye, then b would still win, so that makes no difference in this case. There may be, however, instances of elections where the positioning of byes would affect the outcome of the election.

2.3.7 Copeland's Method

The candidates of an election using Copeland's method (see [BTT89a]) are scored based on victories and defeats in pairwise elections. For each pairwise victory the candidate receives one point, and for each defeat the candidate loses one point. The candidate with the most points wins.

Candidate b defeats both a and c and thus has 2 points. Candidate a defeats c and gets 1 point. Candidate c defeats no others and receives 0 points. Therefore b is the winner of the Copeland election.

Chapter 3

Computational Aspects of Dodgson Elections

This chapter will discuss the wide range of work that has been done by computer scientists on Dodgson elections. Computer scientists have sought a good approximation, among other things, for the Dodgson election because of its relationship with the Condorcet criterion.

3.1 Decision Problems

A decision problem is a question with given inputs that yields a yes or no answer only. It is this class of problems that the major complexity classes P and NP are defined. Decision problems are often defined in two parts, an instance and a question. An instance can be made up of sets, graphs, functions, numbers, etc. and the second part, the question, asks a yes or no question about the instance.

3.1.1 Dodgson score decision problems

Two decision problems related to the Dodgson election problem are the score and winner problems. The score problem determines the score for each candidate. The winner problem returns yes if the input candidate is the winner of the election.

Decision Problem: `DodgsonScore`

Instance: A set C of candidates, a multiset V of votes, $c \in C$ and a positive integer k.
Question Is $Score(C, V, c)$, the Dodgson score of candidate c in the election specified by (C, V), less than or equal to k?

Decision Problem: `DodgsonWinner`
Input: A set C of candidates, a multiset V of votes, $c \in C$.
Question: Is c a winner of the election? That is, does c tie-or-defeat all other candidates in the election?

3.2 Algorithms

An algorithm, simply put, is a means to an end. It is a set of well-defined instructions that achieve some goal. Algorithms can be classified according to the techniques they use to get from input to output (such as greedy or probabilistic algorithms) or they can be classified based on output constraints (such as approximation algorithms). The algorithm presented in chapter 4 is both a greedy algorithm and an approximation algorithm.

3.2.1 Approximation Algorithms

Given an optimization problem (i.e., one that minimizes or maximizes some value) in which each potential solution has a positive cost, we may wish to seek a near-optimal solution due to the time or space necessary to find the optimal solution being too great. We say that an algorithm is a $p(n)$-approximation of an optimization problem where $p(n)$ is called the approximation ratio on input of size n. This ratio means that the cost of the near-optimal solution is within a factor of $p(n)$ of the optimal solution [CLRS01].

For example, let P be an instance of a problem which takes input x. For this instance, assume $|x| = 2$ and the optimal solution is 20. Let algorithm A be a $\ln(|x|)$-approximation of problem P. Algorithm A on input x is guaranteed to output an answer that is within the range of $20 \pm (20)(ln(2))$.

The decision problem `DodgsonScore` is a minimization problem since it seeks the shortest number of adjacent swaps that will make the candidate a Condorcet winner, and thus we can talk about algorithms that approximate it.

The whole purpose of studying approximation algorithms and properties of them is to try to circumvent the amount of effort in terms of time or space complexity one might need to solve a problem.

3.2.2 Self-Knowing Correctness

An algorithm can have the property of self-knowing correctness. For sets S and T and function $f : S \to T$, an algorithm $\mathcal{A} : S \to T \times \{\text{“definitely”}, \text{“maybe”}\}$ is *self-knowingly correct for* f if, for all $s \in S$ and $t \in T$, whenever $\mathcal{A}$ on input s outputs $(t, \text{“definitely”})$ it holds that $f(s) = t$.

3.2.3 Greedy Algorithms

A greedy algorithm takes the best choice at any given moment. It makes locally optimal choices that sometimes yield the globally optimal decision. Greedy algorithms can be used to find exact solutions for some problems and can be used to create approximation algorithms for more complex problems.

Greedy algorithms have the greedy-choice property [CLRS01]. This property says that the globally optimum solution can be arrived at by making locally optimal choices. In order for this property to be satisfied the problem must have an optimal substructure, or a structure which allows it to be broken up into smaller pieces that will allow us to reach the correct answer. For a greedy algorithm that outputs an exact solution, this substructure must lead us to it. For one that gives us an approximation, this substructure must allow us to get an answer within some upper bound of the exact solution.

3.3 Computational Complexity

The study of computational complexity deals with the running time of algorithms in terms of the length of their inputs. This section gives a general definition of some of the complexity classes used in this thesis, for more detail on the concepts of P, NP, NP-hard, and NP-complete, see [GJ79] or [BC93].

3.3.1 Polynomial Time Problems

Let D be a decision problem, x its input and $|x|$ is the length of its its input. If there exists an algorithm A such that A solves D and the running time of A is $O(f(|x|))$ where f is a polynomial function, then D is in P, or the class of polynomial-time computable problems.

3.3.2 NP Problems

NP is the class of decision problems that are solvable by an algorithm that runs in nondeterministic polynomial time. Nondeterminism is the ability of an algorithm to have multiple paths of execution, and guess which path to follow until it arrives at the correct solution. NP can also be described as the class of problems where a solution can be verified in polynomial time (i.e., if we have an output we can check to see if that output is the correct answer in polynomial time). It is clear that P $\subseteq$ NP, but the converse remains an open problem in complexity theory.

3.3.3 Θ_2^P

Θ_2^P is the class of decision problems solvable in polynomial-time with the added benefit of being able to make $O(\log(n))$ (where n is the length of the problem's input) calls to an NP-oracle. An oracle is described as a black box that "magically" knows the answer to its question instantaneously (i.e., as if it ran in constant time).

3.3.4 $\mathcal{C}$-Hard and $\mathcal{C}$-Complete Problems

For complexity class $\mathcal{C}$, a problem is said to be $\mathcal{C}$-hard if it is at least as hard as all other $\mathcal{C}$-complete problems. The easiest way to prove an algorithm is $\mathcal{C}$-hard is to perform a polynomial-time many-one reduction to a problem already known to be $\mathcal{C}$-complete (this covers all other $\mathcal{C}$-complete problems by transitivity). A polynomial-time many-one reduction is a mapping from one problem to another. The mapping must be able to be performed in polynomial time. As an example, the reader can refer to Section 3.6 which illustrates a reduction for the `DodgsonWinner` problem.

A C-complete problem is a problem that is $\mathcal{C}$-hard and is also in the class $\mathcal{C}$. Two problems known to be Θ_2^P-complete include `DodgsonWinner` [HHR97], as defined earlier, and determining the winners of Young elections [RSV03].

3.4 Dodgson and Arrow

Dodgson elections fail to satisfy Arrow's independence of irrelevant alternatives criterion. The following theorem presents a proof of this by counterexample.

Theorem 3. *Dodgson elections do not satisfy IIA.*

Proof. Consider the election where $C = \{a, b, c\}$ and $V = \{18 \text{ of } (b \succ a \succ c), 20 \text{ of } (a \succ c \succ b)\}$. The Dodgson scores of a, b, and c are 0, 4, and 19, respectively, making the preference order output by the election $(a \succ b \succ c)$. Let us now remove the Condorcet winner a to get the election defined by $\hat{C} = \{a, b, c\}$ and $\hat{V} = \{18 \text{ of } (b \succ c), 20 \text{ of } (c \succ b)\}$. In the results of this new election, $(c \succ b)$, which deviates from the results of the original. Therefore Dodgson elections do not satisfy IIA. □

If we restrict our instances of Dodgson$_{IIA}$ (see Chapter 2), then there is a polynomial time algorithm that we can use to find whether or not a particular candidate is a Dodgson winner.

Theorem 4. *There is a polynomial time algorithm for* `DodgsonWinner` *if the input is restricted to instances of Dodgson*$_{IIA}$

Proof. The `DodgsonWinner` problem, when guaranteed to be restricted to instances of Dodgson$_{IIA}$, can be solved in polynomial time via the following algorithm.

```
DodgsonWinner(C, V, c)
1  ▷ C is the set of candidates.
2  ▷ V is the list of votes.
3  ▷ c ∈ C is the candidate which is being decided on as the winner or not
4  for x ∈ C−{c}
5       do V̂ ← RemoveIA(V, {c, x})
6          if IsMajorityWinner({c, x}, V̂, x)
7            then return false
8  return true
```

The function `RemoveIA` modifies the votes in its first argument by removing any candidates not present in its second argument, the irrelevant alternatives. The function `IsMajorityWinner` takes two candidates, a set of votes, and determines which candidate defeats the other in that contest. These functions are clearly computable in polynomial time since the number of candidates is fixed.

As a side note, this algorithm would work for any other IIA-satisfying election system. □

3.5 A Brute Force Algorithm

The following algorithm solves the Dodgson election problem using an exhaustive search for the solution. This algorithm has an exponential running time. It works by first finding every possible sequence of adjacent swaps that would make c a Condorcet winner, and then by returning the length of the shortest sequence. Some shortcutting is done by only making

swaps that will improve c against some other candidate which defeats him, thus a sequence of more than one adjacent swap may be performed in each iteration.

The main algorithm, `DodgsonScore`, determines the candidates in C which defeat c in pairwise elections using the `Pairwise` algorithm. These are the candidates c must advance against. Then, after determining this, it iterates through the votes and determines, for each vote, the length of all possible swap sequences in a particular vote between c and some candidate which defeats it. After building this data set, it calls the `FindMinimumScore` algorithm.

The `FindMinimumScore` algorithm recursively tries all possible swap sequences that were built earlier. It tallies the total length of these sequences and eventually returns the length of the shortest sequence.

`Condorcet` is a trivial algorithm not spelled out in detail here. It takes in candidates C, votes V, and $c \in C$ as inputs and returns true if c is the Condorcet winner of the election system.

```
DodgsonScore(C, V, c)
 1  ▷ C is the set of candidates.
 2  ▷ V is the list of votes.
 3  ▷ c ∈ C is the candidate whose score the algorithm computes.
 4  D ← ∅
 5  for (each d ∈ C−{c})
 6       do if Pairwise(C, V, d, c) > ⌈n/2⌉
 7            then D ← D ∪ {d}
 8  for each vote v ∈ V
 9       do posSwaps[v] ← ∅
10          for (each d ∈ D)
11               do swapnum ← max(0, Position(d, v) − Position(c, v))
12                  if swapnum > 0
13                    then posSwaps[v] ← posSwaps[v] ∪ {swapnum}
14  return FindMinimumScore(C, V, c, posSwaps)
```

```
FindMinimumScore(C, V, c, posSwaps)
 1  if Condorcet(C, V, c)
 2     then return 0
 3  foundvote ← false
 4  for (each v ∈ V)
 5        do if ‖posSwaps[v]‖ > 0 and ¬foundvote
 6              then curvote ← v
 7                   foundvote ← true
 8              else newPosSwaps[v] ← posSwaps[v]
 9  minscore ← ∞
10  if foundvote
11     then for (each s ∈ posSwaps[curvote])
12             do V̂ ← (V−{curvote}) ∪ Swap(c, s,curvote)
13                newscore ← s + FindMinimumScore(C, V̂, c, newPosSwaps)
14                minscore ← min(newscore, minscore)
15  return minscore
```

```
Pairwise(C, V, c, d)
1  winningvotes ← 0
2  for (each v ∈ V)
3       do if Position(c) > Position(d)
4             then winningvotes ← winningvotes+1
5  return winningvotes
```

```
Position(c, v)
1  ▷ Each vote v is treated as an array where v[1] is the least
2  ▷ preferred candidate, v[2] is the second least preferred candidate, and so
3  ▷ on, and v[length[v]] is the most preferred candidate.
4  i ← 1
5  while (i ≤ length[v])
6        do if v[i] = c
7              return i
```

3.6 NP-hardness of `DodgsonWinner`

The proof that `DodgsonWinner` is NP-hard was given in [BTT89b] by constructing a polynomial-time many-one reduction from the problem exact cover by 3-sets. This problem is known to be NP-hard [GJ79]. An instance of the problem is presented here as an example, and the reduction given in [BTT89b] is listed as an example.

3.6.1 Exact Cover by 3-sets

Instance: Set B with $\|B\| = 3q$ and a collection S of 3-element subsets of B.

Question: Does S contain an exact cover for B? In other words, does there exist a subcollection S' of S such that every element of B occurs in exactly one member of S'?

Example of X3C

Let $B = \{a, b, c, d, e, f\}$ and S contains 3-element subsets of B, or

$$S \subseteq \{ \\ \{a, b, c\}, \{a, b, d\}, \{a, b, e\}, \{a, b, f\}, \\ \{a, c, d\}, \{a, c, e\}, \{a, c, f\}, \{a, d, e\},$$

$\{a, d, f\}, \{a, e, f\}, \{b, c, d\}, \{b, c, e\},$
$\{b, c, f\}, \{b, d, e\}, \{b, d, f\}, \{b, e, f\},$
$\{c, d, e\}, \{c, d, f\}, \{c, e, f\}, \{d, e, f\}$

$\}$

The problem would answer "yes" if S were to contain any one of the following sets as a subset, and not otherwise.

1. $\{\{a, b, c\}, \{d, e, f\}\}$
2. $\{\{a, b, d\}, \{c, e, f\}\}$
3. $\{\{a, b, e\}, \{c, d, f\}\}$
4. $\{\{a, b, f\}, \{c, e, f\}\}$
5. $\{\{a, c, d\}, \{b, e, f\}\}$
6. $\{\{a, c, e\}, \{b, d, f\}\}$
7. $\{\{a, c, f\}, \{b, d, e\}\}$
8. $\{\{a, d, e\}, \{b, c, f\}\}$
9. $\{\{a, d, f\}, \{b, c, e\}\}$
10. $\{\{a, e, f\}, \{b, c, d\}\}$

3.6.2 Defining the Candidates

We start with an instance of X3C having a set B and a collection S of 3-subsets of B. The set C of candidates is created by adding two candidates for every element in B. A candidate will also be added for every element in the collection S. A special candidate, called the target candidate, and the candidate that we are trying to find the Dodgson score of, is also making the final number of candidates $2\|B\| + \|S\| + 1$.

3.6.3 Defining the Voters

The set of voters V is made up of three subsets: swing voters, equalizing voters, and incremental voters. The swing voters are built based on the set S' and are how an instance of X3C is transformed into an instance of Dodgson winner. The votes are constructed in such a way that forces all adjacent swaps that will yield the Dodgson score of the target candidate to be made in the swing votes.

The Reduction in Action

Let $B = a, b, c, d, e, f$ and $S = \{s_1 = \{a, b, c\}, s_2 = \{b, d, e\}, s_3 = \{c, e, f\}, s_4 = \{d, e, f\}\}$. First we need to create the set of candidates C. We will add two candidates for each member of B, one candidate for each member of the set S and our target candidate t making

$$C = \{a, b, c, d, e, f, \hat{a}, \hat{b}, \hat{c}, \hat{d}, \hat{e}, \hat{f}, s_1, s_2, s_3, s_4, t\}.$$

Now we will create swing voters, V_s, with one vote for each member of S. The set V_s will contain the following four votes. Note that the first three candidates of each vote correspond to the elements of the sets in S.

1. $(a \succ b \succ c \succ s_1 \succ t \succ d \succ e \succ f \succ \hat{a} \succ \hat{b} \succ \hat{c} \succ \hat{d} \succ \hat{e} \succ \hat{f} \succ s_2 \succ s_3 \succ s_4)$
2. $(b \succ d \succ e \succ s_2 \succ t \succ a \succ c \succ f \succ \hat{a} \succ \hat{b} \succ \hat{c} \succ \hat{d} \succ \hat{e} \succ \hat{f} \succ s_2 \succ s_3 \succ s_4)$
3. $(c \succ e \succ f \succ s_3 \succ t \succ a \succ b \succ d \succ \hat{a} \succ \hat{b} \succ \hat{c} \succ \hat{d} \succ \hat{e} \succ \hat{f} \succ s_2 \succ s_3 \succ s_4)$
4. $(d \succ e \succ f \succ s_4 \succ t \succ a \succ b \succ c \succ \hat{a} \succ \hat{b} \succ \hat{c} \succ \hat{d} \succ \hat{e} \succ \hat{f} \succ s_2 \succ s_3 \succ s_4)$

In the following table, each candidate is given with the number of votes in V_s that it defeats candidate t in.

candidate	votes	candidate	votes	candidate	votes	candidate	votes
a	1	e	3	$\hat{c}$	0	s_1	1
b	2	f	2	$\hat{d}$	0	s_2	1
c	2	$\hat{a}$	0	$\hat{e}$	0	s_3	1
d	2	$\hat{b}$	0	$\hat{f}$	0	s_4	1

Next we will have to create the equalizing voters, V_e using the above table. The maximum number of votes any candidate from B has over t is 3 in this example. Given that, the votes of the set V_e when constructed are

1. $(a \succ \hat{a} \succ t \succ b \succ c \succ d \succ e \succ f \succ \hat{b} \succ \hat{c} \succ \hat{d} \succ \hat{e} \succ \hat{f} \succ s_1 \succ s_2 \succ s_3 \succ s_4)$
2. $(a \succ \hat{a} \succ t \succ b \succ c \succ d \succ e \succ f \succ \hat{b} \succ \hat{c} \succ \hat{d} \succ \hat{e} \succ \hat{f} \succ s_1 \succ s_2 \succ s_3 \succ s_4)$
3. $(b \succ \hat{b} \succ t \succ a \succ c \succ d \succ e \succ f \succ \hat{a} \succ \hat{c} \succ \hat{d} \succ \hat{e} \succ \hat{f} \succ s_1 \succ s_2 \succ s_3 \succ s_4)$
4. $(c \succ \hat{c} \succ t \succ a \succ b \succ d \succ e \succ f \succ \hat{a} \succ \hat{b} \succ \hat{d} \succ \hat{e} \succ \hat{f} \succ s_1 \succ s_2 \succ s_3 \succ s_4)$
5. $(d \succ \hat{d} \succ t \succ a \succ b \succ c \succ e \succ f \succ \hat{a} \succ \hat{b} \succ \hat{c} \succ \hat{e} \succ \hat{f} \succ s_1 \succ s_2 \succ s_3 \succ s_4)$
6. $(f \succ \hat{f} \succ t \succ a \succ b \succ c \succ d \succ e \succ \hat{a} \succ \hat{b} \succ \hat{c} \succ \hat{d} \succ \hat{e} \succ s_1 \succ s_2 \succ s_3 \succ s_4)$

We will now re-compute the results of the pairwise elections against t using $V_s \cup V_e$.

candidate	votes	candidate	votes	candidate	votes	candidate	votes
a	3	e	3	$\hat{c}$	1	s_1	1
b	3	f	3	$\hat{d}$	1	s_2	1
c	3	$\hat{a}$	2	$\hat{e}$	0	s_3	1
d	3	$\hat{b}$	1	$\hat{f}$	1	s_4	1

The last set of votes are called incremental votes and will be placed in the set V_i. Their purpose is to make each candidate from B defeat t by exactly one vote. Currently, t defeats

all members of B 7-to-3 and therefore five incremental votes will be added making all five members of B defeat t 8-to-7. These votes will be

1. $(a \succ b \succ c \succ d \succ e \succ f \succ \hat{a} \succ \hat{b} \succ \hat{c} \succ \hat{d} \succ \hat{e} \succ \hat{f} \succ t \succ s_1 \succ s_2 \succ s_3 \succ s_4)$
2. $(a \succ b \succ c \succ d \succ e \succ f \succ \hat{a} \succ \hat{b} \succ \hat{c} \succ \hat{d} \succ \hat{e} \succ \hat{f} \succ t \succ s_1 \succ s_2 \succ s_3 \succ s_4)$
3. $(a \succ b \succ c \succ d \succ e \succ f \succ \hat{a} \succ \hat{b} \succ \hat{c} \succ \hat{d} \succ \hat{e} \succ \hat{f} \succ t \succ s_1 \succ s_2 \succ s_3 \succ s_4)$
4. $(a \succ b \succ c \succ d \succ e \succ f \succ \hat{a} \succ \hat{b} \succ \hat{c} \succ \hat{d} \succ \hat{e} \succ \hat{f} \succ t \succ s_1 \succ s_2 \succ s_3 \succ s_4)$
5. $(a \succ b \succ c \succ d \succ e \succ f \succ \hat{a} \succ \hat{b} \succ \hat{c} \succ \hat{d} \succ \hat{e} \succ \hat{f} \succ t \succ s_1 \succ s_2 \succ s_3 \succ s_4)$

Now we can set the votes of this election: $V = V_s \cup V_e \cup V_i$. The final tabulation of pairwise elections versus candidate t is given in the following table.

candidate	votes	candidate	votes	candidate	votes	candidate	votes
a	8	e	8	$\hat{c}$	6	s_1	1
b	8	f	8	$\hat{d}$	6	s_2	1
c	8	$\hat{a}$	7	$\hat{e}$	5	s_3	1
d	8	$\hat{b}$	6	$\hat{f}$	6	s_4	1

Now that the election system is complete we will set the integer k, defined in the Dodgson score decision problem, to be $\frac{4\|B\|}{3}$, or, in this case, 8. For the reduction to be valid, we must be able to use the instance of X3C to solve the instance of `DodgsonScore` and vice versa.

To find out whether the Dodgson score of t is less than or equal to 8 we will need to solve X3C on inputs B and S. We are forced to perform all switches using the swing votes. If t were to use a swing vote it must make 4 switches in order to defeat every candidate, therefore, the 4 switches need to be made, in this example, in exactly 2 of the swing votes. The reader will recall that each swing vote corresponds to a member of the collection S.

The members of the exact 3-cover of B using S will correspond to the votes where the switches need to be made. In this example $S' = \{\{a, b, c\}, \{d, e, f\}\}$ and therefore the two votes that need to have switches made are

1. $(a \succ b \succ c \succ s_1 \succ t \succ d \succ e \succ f \succ \hat{a} \succ \hat{b} \succ \hat{c} \succ \hat{d} \succ \hat{e} \succ \hat{f} \succ s_2 \succ s_3 \succ s_4)$
2. $(d \succ e \succ f \succ s_4 \succ t \succ a \succ b \succ c \succ \hat{a} \succ \hat{b} \succ \hat{c} \succ \hat{d} \succ \hat{e} \succ \hat{f} \succ s_2 \succ s_3 \succ s_4)$

To go the other way we would solve the `DodgsonScore` problem but along the way track the votes in which we made adjacent swaps. These votes correspond to the sets in S which would solve the instance of X3C.

It is this section which inspired the algorithm found in chapter 4. The reader can see from the reduction that a greedy swap is never available between the target candidate and candidates that defeat him. It is the investigation of the availability, or lack thereof, these swaps which are the basis of the our approximation algorithm.

3.7 Greedily Computing the Score

Most of the difficulty of computing the Dodgson winner comes from the immense amount of backtracking required. Since it is a minimization problem (find the minimum number of swaps required), all possible ways of swapping a candidate with its adjacent alternatives must be tried in every single vote (unless of course, the candidate is a Condorcet winner). Homan and Hemaspaandra found that a good heuristic can be obtained by halting the backtracking altogether and just take swaps that are known to help the candidate in question. Further, their algorithm was frequently correct at calculating the exact Dodgson score. In order to prove that the algorithm they developed, called `GreedyWinner`, was frequently correct they introduced the notion of self-knowing correctness.

The reader may recall from section 3.2.1 that a self-knowingly correct algorithm A' of a problem X is one that, along with the result, also gives, what is called, confidence level from the set $\{definitely, maybe\}$. If the confidence level is $definitely$ then the output of A'

is also the correct answer. If the confidence level is $maybe$ then the output of A' may or may not be correct.

In `GreedWinner` the confidence level depends on the confidence level of each `GreedyScore` computation completed. The confidence level of each `GreedyScore` computation is depends on the notion of a vote deficit. The deficit of a voter c from voter d is the maximum of zero or the number of times d defeats c in a pairwise election minus the number of times c defeats d in a pairwise election. If the deficit of c and every voter in $C - \{c\}$ is 0, then c is the Condorcet winner.

The algorithm is simple. It iterates through all the votes and determines for each candidate other than c the number of votes in which c has a greedy swap available with them. No swaps are actually made and only one pass through the votes is done. If for each candidate other than c there are enough greedy swaps available to overtake the deficit between the candidate and c then the deficit is the Dodgson score and the algorithm yields a confidence of $definitely$, otherwise, the total deficit is returned with a confidence of $maybe$.

3.7.1 `GreedyScore` and `GreedyWinner`

It was shown in [HH06] that the `GreedyScore` algorithm is not only fast and self-knowingly correct, but also very frequently correct when the voters greatly outnumber the candidates. The algorithm outputs $maybe$ less than $2(m^2 - m)e^{\frac{-n}{8m^2}}$ of the time over a uniform distribution of election systems, where m is the number of candidates and n is the number of votes. The probability of the algorithm outputting $maybe$ shrinks asymptotically as we increase the number of voters over the number of candidates.

Chapter 4

Marginal Cost Dodgson Election Algorithms

Marginal Cost Dodgson is an algorithm which approximates the Dodgson score of a candidate. It builds upon the techniques used by `GreedyScore` and is itself a greedy algorithm. The algorithm also retains the self-knowingly correct property. The algorithm works by, for each vote, computing the most efficient swap, or the swap with the least marginal cost associated with it. This swap is the one that will yield the highest deficit reduction for the number of swaps taken. After computing the marginal cost for each swap, the algorithm then makes the swap with the highest efficiency. The marginal costs and swaps will then be recalculated using the altered preference profile and this process continues until the candidate is the Condorcet winner.

4.1 Building Blocks

The foundation of this algorithm lies in two things: deficits and edits. A deficit is defined as the number of votes candidate a needs over candidate b in order for a to defeat b in a pairwise election. An edit is a swap, or the exchanging of places between two adjacent candidates in a vote. We define the two terms more formally below.

Definition 5. *For every pair of distinct candidates $c, d \in C$ and every preference profile*

$P = \langle >_1, \ldots, >_n \rangle$, *c's vote deficit in* P *with* d *is* $Deficit(P, c, d) = \max\{0, ||\{i \in V \mid d >_i c\}|| - ||\{i \in V \mid c >_i d\}||\}$. *The* total deficit *of* c *is*

$$Deficit(P, c) = \sum_{d \in C - \{c\}} Deficit(P, c, d).$$

Thus c *is a* Condorcet winner *if and only if* $Deficit(P, c) = 0$.

$Deficit(P, c)$ is sometimes known as the Tideman score [Tid87], which forms the basis of the Tideman (a.k.a. ranked pairs) voting rule.

Definition 6. *An edit is a mapping* $e : \bigcup_{i=0}^{\infty} \mathcal{L}(C)^i \to \bigcup_{i=0}^{\infty} \mathcal{L}(C)^i$. *Let* $P \circ e$ *denote the application of* e *to some preference profile* P. *A sequence of edits* $\langle e_1, \ldots, e_p \rangle$ *is called a Condorcet sequence if* $Deficit(P \circ e_1 \circ \cdots \circ e_p, c) = 0$

A swap is an edit, designated by an ordered pair $(i, j) \in \mathbb{N}^2$, *that takes a preference profile* $P = \langle >_1, \ldots, >_n \rangle$ *and outputs* $\langle >'_1, \ldots, >'_n \rangle$, *which is just like* P *except that, if* $1 \leq i \leq n$ *and* $0 < j < m$, *then for* $c, d \in C$ *satisfying* $d(>_i) = j = c(>_i) + 1$ *it holds that* $c(>'_i) = j = d(>'_i) + 1$. *This implies that* c, d *are adjacent in both rankings,* $d >_i c$, *and* $c >'_i d$. *Candidates* c *and* d *are said to be involved in the swap.*

Now that we have defined both deficits and edits formally, we can define a mechanism to reduce the deficit. The very heart of the Dodgson election system is to find the sequence of edits that makes the target candidate a Condorcet winner and is minimal. Therefore we only make swaps that reduce the deficit against other candidates so maximal efficiency is obtained, if not the exact solution itself.

Definition 7. *A deficit reduction is a 4-tuple* (P, c, e, d) *where* P *is a preference profile,* c *and* d *are candidates, and* e *is an edit such that* $Deficit(P, c, d) > Deficit(P \circ e, c, d)$. *The full sequence of deficit reductions with respect to candidate* c *over a sequence of edits* $\langle e_1, \ldots, e_p \rangle$ *on a preference profile* P, *denoted* $D(P, c, \langle e_1, \ldots, e_p \rangle)$, *is the non-repeating sequence of deficit reductions* $\langle (P_1, e_{i_1}, c, d_1), \ldots, (P_q, e_{i_q}, c, d_q) \rangle$ *of maximum length such that, for all* $k \in \{1, \ldots, q\}$, $P_k = P \circ e_{i_j} \circ \cdots \circ e_{i_{k-1}}$, $Deficit(P_k, c, d_k) > Deficit(P_k \circ e_{i_k}, c, d_k)$, *and for all* $j \in \{1, \ldots, k-1\}$, $i_j \leq i_k$. *In more simpler terms, this is the*

sequence of swaps in which candidate c uses to overcome candidate d_k in the election. Each swap made in the sequence should reduce the deficit between them.

From these definitions, we can define the Dodgson election voting rule in terms of them. Let $\mathcal{S}$ be the collection of all sequences of swaps. The Dodgson score of candidate c in profile P is the smallest $p \in \mathbb{N}$ such that

$$(\exists \langle e_1, \ldots, e_p \rangle \in \mathcal{S})[\mathit{Deficit}(P \circ e_1 \circ \cdots \circ e_p, c) = 0].$$

4.2 The Marginal Cost Dodgson Algorithm

Below is the algorithm, emphasizing key components and omitting important but mundane steps. For instance, the algorithm needs to compute $\mathit{Deficit}(P, c)$. The details of the algorithm will be discussed in the next section. Here, $\mathcal{E}$ is a collection of sequences of swaps. The variable $\mathcal{E}$ is implemented as a priority queue, where priority is given to sequences of edits S' that, when applied to the preference profile P, have the fewest edits per deficit reduction, i.e., that minimize $|S'|/|D(P, c, S')|$. We call this quantity the marginal cost of S' because it is cost in terms of swaps for making that deficit reduction. We define $|S'|/|D(P, c, S')| = \infty$ whenever $|D(P, c, S')| = 0$.

For example, Let $C = a, b, c, d, e, f$, the votes V are arbitrary. Assume c is defeated by b, d, and e in pairwise elections using the votes in V. In the vote $(a \succ b \succ d \succ e \succ f \succ c)$, to move candidate c to the top would 5 swaps and overtake 2 live candidates, therefore the marginal cost of this move is 5 / 2 or 2.5. Greedy swaps always have a marginal cost of 1 and will be performed first in the algorithm, since 1 is the lowest marginal cost possible.

S is a list of edits made so far in the algorithm, and let P be a preference profile and c be a target candidate in the election. The algorithm continues while c is not a Condorcet winner (meaning that $\mathit{Deficit}(P, c)$, or the total deficit of c against all other candidates, is greater than 0). For each iteration it chooses the sequence of edits with the least marginal cost, and if tied for marginal cost, the lowest number of edits, and stores that sequence into S'. It then changes the preference profile P to reflect the edits in S'. Finally, in the loop,

the edits stored in S' are added to S. When the algorithm is finished, it returns the length of S along with a confidence level. If swaps were used that had marginal costs other than 1, then the confidence level will be $maybe$, otherwise it will be $definitely$.

`MCDodgsonScore`(P, c)

1 $S = \emptyset$
2 $confidence =$"definitely"
3 **while** $Deficit(P, c) > 0$
4 **do** $S' = \text{argmin}_{S'' \in \mathcal{E}} |S''|/|D(P, c, S'')|$
5 **if** $|S'|/|D(P, c, S')| \neq 1$
6 **then** $confidence =$"maybe"
7 $\{e_1, \ldots, e_p\} = S'$
8 $P = P \circ e_1 \circ \cdots \circ e_p$
9 $concatenate(S, \{e_1, \ldots, e_p\})$
10 **return** $(|S|, confidence)$

Let P be a preference profile and let $\mathcal{E}'$ be the collection of all swap sequences where, for each sequence, there is a single voter's preference list to which all swaps in the sequence apply. Note then that every such sequence has a distinct last element, so we can represent each sequence in $\mathcal{E}'$ by storing its last element only. Let us call the voting system based on the generic algorithm with $\mathcal{E} = \mathcal{E}'$ "Marginal-Cost-Greedy-Dodgson." The winners are, as in the Dodgson election, the candidates with the lowest scores.

The above algorithm is both efficient and very frequently correct because it is at least as frequently correct as GreedyWinner and has a running time of $O(N^2 \log N)$

Theorem 8. *The running time of Marginal-Cost-Greedy-Dodgson, when $\mathcal{E} = \mathcal{E}'$, is $O(N^2 \log N)$, where N is the length of the input.*

Proof. Let (P, c) be the input to the algorithm, where $\mathcal{E} = \mathcal{E}'$ and P has m candidates and n voters. We first need to initialize the data structures used. It takes linear time to calculate $Deficit(P, c, d)$ on all $d \in C - \{c\}$ (note that we can compute $Deficit(P, c)$ at the same

time). Next we need to initialize $\mathcal{E}'$. There are at most $n(m-1)$ sequences S' in $\mathcal{E}'$, and there are at most $m(m-1)/2$ distinct values for $|S'|/|D(P,c,S')|$ that any such sequence can take. So (regarding $\mathcal{E}$ as a priority queue) it takes $O(\log m)$ comparisons to add any such sequence (which we recall is represented by the last element of the sequence) to $\mathcal{E}$. Note that we can calculate $|S'|/|D(S')|$ for every sequence S' in $\mathcal{E}$ in a single pass through P. The worst case is when n is as small as possible, so the worst case running time for initialization is $O(N \log N)$

After initialization, the algorithm performs swaps on P until c is the Condorcet winner. Note that any given swap is performed at most once. For each swap applied, the algorithm must remove the corresponding swap from the queue (since whenever a swap is applied it follows that the swap sequence ending with that swap has also been applied), and it must update the marginal cost of each swap sequence remaining in $\mathcal{E}$ that applies to the current voter's preferences. Thus, every swap may require $O(m)$ updates to $\mathcal{E}$. Assuming that all swaps in $\mathcal{E}$ sharing a common voter are connected via a linked list, each update can happen in constant time. As during initialization, the worst case for these procedures occurs when n is as small as possible, so the running time for this part of the algorithm is $O(N^2)$

Finally, every time a swap causes the deficit against some opponent to go from positive to zero the entire queue needs to be reprioritized, which means we must pass through all swap sequences and recalculate This can happen at most $(m-1)$ times. Again, the worst-case running time is when n is as small as possible, so it is $O(N^2 \log N)$.

□

Theorem 9. *If* `GreedyWinner` *outputs* $definitely$ *as part of its results, then* `MCDodgsonWinner` *also outputs "definitely" as part of its result.*

Proof. Let (C, V, c) be a Dodgson triple where `GreedyWinner` outputs $definitely$. If that is the case, then there must be enough greedy swaps in V with respect to c and the opponents that are defeating him in order to cover his deficit against each of them. in `MCDodgson` a vote cannot have efficiency greater than one (one swap, for one gain, three swaps, for three gain, etc.) and therefore all greedy swaps will be computed first. In (C, V, c) we know

that there are enough greedy swaps to cover the deficit because `GreedyWinner` outputs "definitely" as part of its result, therefore `MCDodgsonWinner` will also output *definitely*. □

4.3 MCDodgson's Approximation Bound

We turn now to the approximation bound. Our proof assumes there is a Condorcet sequence of swaps witnessing the Dodgson score of c. The following proposition shows that our assumption is valid. Here, a Condorcet swap sequence means a sequence of swaps that would make a candidate a Condorcet winner.

Proposition 10. *For every preference profile P and candidate c there is a Condorcet swap sequence of length equal to the Dodgson score of c.*

Proof. Let p be the Dodgson score of c and $\langle s_1, \ldots, s_p \rangle$ be a Condorcet swap sequence with respect to candidate c on preference profile $P = \langle >_1, \ldots, >_n \rangle$. Let $\langle >'_1, \ldots, >'_n \rangle = P \circ s_1 \circ \cdots \circ s_p$. Let S be as in the algorithm. Choose $i \in V$ and let $\langle s'_1, \ldots, s'_q \rangle$ be the subsequence of $\langle s_1, \ldots, s_p \rangle$ consisting of all swaps on voter i's preferences. Let $d' = \operatorname{argmax}_{d \in C : c >'_i d}(d(>_i) - c(>_i))$. Since it requires at least $d'(>_i) - c(>_i)$ swaps in order for $c >'_i d'$ to hold, it must be the case that $|\langle s'_1, \ldots, s'_q \rangle| \geq d'(>_i) - c(>_i)$. So, removing from S each swap in $\langle s'_1, \ldots, s'_q \rangle$ and appending the sequence $\langle (i, c(>_i) + 1), \ldots, (i, d'(>_i)) \rangle$ yields a Condorcet sequence that has no more swaps than S originally had. □

Theorem 11. *Marginal-Cost-Greedy-Dodgson is an $(\ln m + 1)$-approximation of Dodgson score, where m is the number of candidates in the input election.*

Proof. Let P be a preference profile over m candidates and n voters and let c be a candidate in $\{1, \ldots, m\}$. Let x be the Dodgson score of c on P and let S^* be a c-normal Condorcet sequence of P. Let $y = \mathit{Deficit}(P, c)$ and let $\langle (P^*_1, c, s^*_1, d^*_1), \ldots, (P^*_y, c, s^*_y, d^*_y) \rangle = D(P, c, S^*)$. Let S be the same as in the algorithm on input (P, c) at the time line 11 is

reached (i.e., it is the sequence of all swaps the algorithm applies to P), and let $\langle (P_1, c, s_1, d_1), \ldots, (P_y, c, s_y, d_y) \rangle = D(P, c, S)$.

The basic idea behind our proof is that the number of deficit reductions in a sequence that witnesses the Dodgson score of c, such as S^*, is equal to the number of deficit reductions in the sequence S that the algorithm produces. So to compare $|S|$ to $|S^*|$ we partition the swaps in S (respectively, S^*) among the deficit reductions and then match the deficit reductions in S to those in S^*. The partitioning is easy: For S it is just the marginal cost associated with each deficit reduction. For S^* we fudge the marginal cost in a straightforward way. The matching and the order in which matched elements are compared are the trickiest parts of the proof.

For every $k \in \{1, \ldots, y\}$, let $r(s_k)$ denote the marginal cost the algorithm associates with s_k (i.e., $|S'|/|D(P, c, S')|$, where S' and P are as in line 5 during the iteration when the algorithm chooses s_k to be in S'). Clearly,

$$|S| = \sum_{k=1}^{y} r(s_k).$$

Let σ denote a permutation over $\{1, \ldots, y\}$ that satisfies the following constraints.

1. For every $j \in \{1, \ldots, y\}$, $d^*_j = d_{\sigma(j)}$.

2. For every $j, k \in \{1, \ldots, y\}$, if $s^*_k = s_j$ then $k = \sigma(j)$.

Clearly, such a mapping exists.

For each $i \in \{1, \ldots, n\}$, let S^*_i (respectively, D^*_i) be the subsequence of all swaps in S^* (respectively, $\langle s^*_1, \ldots, s^*_y \rangle$) that apply to voter i only (i.e., all swaps that for some j are of the form (i, j)). Let $p = |D^*_i|$ and let $D_i = \langle s_{k_1}, \ldots, s_{k_p} \rangle$ be the subsequence of all swaps in $\langle s_1, \ldots, s_y \rangle$ that σ maps to some element in D^*_i. In particular, this subsequence preserves the order in which the algorithm applies the swaps.

We claim, for every $q \in \{1, \ldots, p\}$, that $r(s_{k_q}) \leq |S^*_i|/(|D^*_i| + 1 - q)$. This is because, by our construction of σ, at the time the algorithm is about to choose s_{k_q} it has not chosen $s^*_{\sigma(k_q)}$ nor any of the other swaps in S^*_i that come after it (in fact, the algorithm may not

have chosen a single swap in S_i^*). Because the subsequence $\langle s_{k_1}, \ldots, s_{k_p} \rangle$ preserves the order in which the swaps were made, the algorithm still needs at this point to close deficits against the candidates $d_{k_q}, d_{k_q+1}, \ldots, d_{k_p}$ $(= d^*_{\sigma(k_q)}, d^*_{\sigma(k_q+1)}, \ldots, d^*_{\sigma(k_p)})$.

So at the time the algorithm chooses swap s_{k_q}, it could instead take the longest subsequence of S_i^* that remains unchosen. Obviously, this subsequence is at most $|S_i^*|$ swaps long and, as discussed above, it yields at least $|D_i^*| + 1 - q$ deficit reductions. Since s_{k_q} was chosen as part of a sequence S' for which $|S'|/|D(P, S', c)|$ $(= r(s_{k_q})$, where P here is taken to be in the same state as when S' was chosen) was as small as possible, our claim holds. But then

$$\begin{aligned} |S| &= \sum_{k=i}^{y} r(s_k) \\ &\leq \sum_{i=1}^{n} \sum_{q=1}^{|D_i^*|} |S_i^*|/(|D_i^*| + 1 - q) \\ &\leq \sum_{i=1}^{n} \sum_{q=1}^{m} |S_i^*|/(m + 1 - q) \\ &\leq |S^*| \ln m + 1 \end{aligned}$$

□

4.4 MCDodgson and Dodgson

Now that we know this is a good approximation of Dodgson elections, we wish to see which properties of the original system it retains.

Theorem 12. *Marginal-Cost-Greedy-Dodgson satisfies the Condorcet criterion*

Proof. Initially $S = \emptyset$, and new elements will only be added to S if the while loop that is the heart of the algorithm executes. If c is a Condorcet winner, then the deficit against any other candidate will always be 0 by the definitions of Condorcet winner and deficit. Thus, the while loop never executes and S remains $\emptyset$. Therefore a score of zero is returned and the algorithm recognizes the candidate as a winner.

□

Theorem 13. *There exists a preference profile with a candidate c such that c is a Marginal-Cost-Greedy-Dodgson winner but c is not a Dodgson winner.*

Proof. Let $C = \{a, b, c, d\}$ and let V have four votes:

1. $(d \succ b \succ c \succ a)$
2. $(a \succ b \succ c \succ d)$
3. $(b \succ c \succ d \succ a)$
4. $(a \succ d \succ b \succ c)$

The Dodgson scores of a, b, c, d are 3, 5, 2, and 3 respectively. The Marginal-Cost-Greedy-Dodgson scores of the candidates are 2, 4, 2, and 3 respectively. This means that candidates a and c are Marginal-Cost-Greedy-Dodgson winners, but only c is the Dodgson winner. □

Theorem 14. *There exists a preference profile with a candidate c such that c is a Dodgson winner but c is not a Marginal-Cost-Greedy-Dodgson winner.*

Proof. Let $C = \{a, b, c, d\}$ and let V have four votes:

1. $(c \succ a \succ d \succ b)$
2. $(b \succ a \succ d \succ c)$
3. $(d \succ b \succ a \succ c)$
4. $(c \succ d \succ a \succ b)$

The Dodgson scores of a, b, c, d are 3, 2, 3, and 4 respectively. The Marginal-Cost-Greedy-Dodgson scores of the candidates are 4, 3, 2, and 4 respectively. This means that candidate b is the only Dodgson winner and the only Marginal-Cost-Greedy-Dodgson winner is candidate c. □

4.5 Detail of the Winner and Score Algorithms

Although the winner algorithm as presented earlier is useful in proving theorems it may not be as useful in implementation. Therefore we present a more detailed version of the algorithm (including its score component) that a programmer may find easier to translate into the language of his choice.

Before the actual algorithm, we must define some data structures, types, and simple functions. Let S be a collection of swaps, and E be a priority queue where swaps of least marginal cost are considered to have highest priority. Swaps of the same marginal cost are placed in arbitrary order. We can acces the members of E, along with their priority, by the notation $E[i]$ which represents the swap in E at position i. Thus $E[0]$ would be the member of highest priority (or in this case, least marginal cost) along with its marginal cost. Let $v[i]$ denote the candidate who is in position i in vote v. $Position(c, v)$ is a simple polynomial-time function which returns an integer representing the position of candidate c in vote v. Let $Swap(c, d, v)$ be an object representing a swap between candidates c and d in vote v. Finally, let $deficit(c, V)$ denote the total deficit of candidate c against other candidates in the collection of votes V, this is clearly a polynomial-time operation.

The algorithm works by first initializing S to an empty collection of swaps. The main while loop executes until c no longer has a deficit and is therefore a Condorcet winner. The main while loop fills the priority queue by computing the marginal cost of all possible swaps c could make with other candidates. After all marginal costs are computed, only the swap with the lowest marginal cost is used. That swap is added to our collection of swaps S, and finally the swap s is carried out on the votes V before the loop returns to the top where the new, modified version of the votes is used in the computations of the next iteration.

If we never encounter a marginal cost other than 1, then we know for sure that the score computed is definitely the Dodgson score. The total length of all the swaps in S is computed, and then that, along with the confidence measure, is returned.

```
MCDodgsonScore(C, V, c)
 1  S ← ∅
 2  confidence ← "definitely"
 3  while deficit(c, V) > 0
 4       do E ← ∅
 5          for each v in V
 6          pos ← Position(c, v)
 7               do D ← candidates that defeat c in pairwise elections
 8                  counter ← 0
 9                  for each i in [1, pos) iterated in reverse order
10                       do d ← v[i]
11                          if d ∈ D
12                            then counter ← counter + 1
13                                 marginalcost ← (pos − i)/counter
14                                 Add Swap(c, d, v) to E with priority marginalcost
15          (s, marginalcost) ← E[0]
16          if marginalcost ≠ 1
17            then confidence ← "maybe"
18          Add s to S
19          V ← V after making swap s
20  length ← 0
21  for each s in S
22       do length ← length+ the distance between the candidates in swap s
23  return (length, confidence)
```

The algorithm `MCDodgsonWinner` simply determines whether the candidate c is the winner of the Dodgson Election using the `MCDodgsonScore` algorithm as its scoring mechanism. A candidate is declared a winner if no other candidate has a score greater than his. Ties are acceptable.

```
MCDodgsonWinner(C, V, c)
1  cscore ← MCDodgsonScore(C, V, c)
2  for (each d ∈ C−{c})
3       do if MCDodgsonScore(C, V, d) > cscore
4            then return False
5  return True
```

Chapter 5

Conclusion and Future Research

In this thesis we discussed an approximation for scoring Dodgson elections which uses and improves upon the technique of the `GreedyScore` algorithm created by Homan and Hemaspaandra. Such an approximation algorithm is necessary because the Dodgson score decision problem falls in the complexity class of NP.

The marginal cost algorithm for computing Dodgson scores is a polynomial-time approximation. The algorithm is also among the best approximations available for the problem since it is an $(\ln m + 1)$-approximation of Dodgson score, where m is the number of candidates in the input election. It was noted that $(\ln m + 1)$ is a lower bound on the error an approximation for Dodgson score can have in [CCF$^+$09] using an observation from McCabe-Dansted's thesis [MD06].

This marginal algorithm uses a general framework which could be extended to include other edit-based election systems (such as Young elections). The algorithm is self-knowingly correct, but little is known about its exact frequency of correctness other than that it is at least as frequently correct as the `GreedyScore` algorithm. The question of the exact frequency of correctness remains open for further research.

Bibliography

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

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

[Bla58] D. Black. *The Theory of Committees and Elections*. Cambridge University Press, 1958.

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

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

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

[CCF+09] I. Caragiannis, J. Covey, M. Feldman, C. Homan, C. Kaklamanis, N. Karanikolask, A. Procaccia, and J. Rosenschein. On the approximability of dodgson and young elections. In *Proceedings of the 20th Annual Symposium of Discrete Algorithms*, 2009.

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

[Con85] M. J. A. N. de Caritat, Marquis de Condorcet. *Essai sur l'Application de L'Analyse à la Probabilité des Décisions Rendues à la Pluralité des Voix*. 1785. Facsimile reprint of original published in Paris, 1972, by the Imprimerie Royale.

[CT65] C. Campbell and G. Tullock. A measure of importance of cyclical majorities. *The Economic Journal*, 75:853–857, 1965.

[dB81] J. C. de Borda. Mémoire sur les élections au scrutin. *Histoire de l'Académie Royale des Sciences*, 1781.

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

[Dod76] C. Dodgson. A method of taking votes on more than two issues. Pamphlet printed by the Clarendon Press, Oxford, and headed "not yet published" (see the discussions in [MU95, Bla58], both of which reprint this paper), 1876.

[EL06] E. Elkind and H. Lipmaa. Hybrid voting protocols and hardness of manipulation. In *First International Workshop on Computational Social Choice*, December 2006.

[ER91] E. Ephrati and J. Rosenschein. The Clarke tax as a consensus mechanism among automated agents. In *Proceedings of the 9th National Conference on Artificial Intelligence*, pages 173–178. AAAI Press, 1991.

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

[FKS03] R. Fagin, R. Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 301–312. ACM Press, 2003.

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

[HH06] C. Homan and L. Hemaspaandra. Guarantees for the success frequency of an algorithm for finding Dodgson-election winners. In *Proceedings of the 31st International Symposium on Mathematical Foundations of Computer Science*, pages 528–539. Springer-Verlag *Lecture Notes in Computer Science #4162*, August/September 2006.

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

[MD06] J. McCabe-Dansted. Computational feasibility of Dodgson's rule. *Master's Thesis*, 2006.

[MDPS06] J. C. McCabe-Dansted, G. Pritchard, and A. Slinko. Approximability of Dodgson's rule. In *First International Workshop on Computational Social Choice*, December 2006.

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

[PHG00] D. Pennock, E. Horvitz, and C. Giles. Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering. In *Proceedings of the 17th National Conference on Artificial Intelligence*, pages 729–734. AAAI Press, 2000.

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

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

[You77] H. Young. Extending condorcet's rule. *Journal of Economic Theory*, 16:335–353, 1977.