2010

Manipulation of elections by minimal coalitions

Christopher Connett

Manipulation of Elections by Minimal Coalitions

Master of Science Thesis

Christopher Jay Connett
Rochester Institute of Technology
B. Thomas Golisano College of Computing and Information Sciences
Department of Computer Science
102 Lomb Memorial Drive
Rochester, New York 14623
USA

June 18, 2010

Signatures

I, Christopher Jay Connett, submit this thesis in partial fulfillment of the requirements for the degree of Master of Science in Computer Science. It is approved by the committee members below.

Christopher Jay Connett

Edith Hemaspaandra, Ph. D.
Advisor and Chair

Christopher Homan, Ph. D.
Reader

Ivona Bezáková, Ph. D.
Observer

Abstract

Social choice is the study of the issues arising when a population of individuals attempts to combine its views with the objective of determining a collective policy. Recent research in artificial intelligence raises concerns of artificial intelligence agents applying computational resources to attack an election. If we think of voting as a way to combine honest preferences, it would be undesirable for some voters cast ballots that differ from their true preferences and achieve a better result for themselves at the expense of the general social welfare. Such an attack is called manipulation. The Gibbard-Satterthwaite theorem holds that all reasonable voting rules will admit a situation in which some voter achieves a better result for itself by misrepresenting its preferences. Bartholdi and Orlin showed that finding a beneficial manipulation under the single transferable vote rule is NP-Complete. Our work explores the practical difficulty of the coalitional manipulation problem. We computed the minimum sizes of successful manipulating coalitions, and compared this to theoretical results.

Acknowledgments

This research was supported in part by grant NSF-IIS-0713061.

Contents

List of Figures

Chapter 1

Introduction

Social choice is an academic field concerned with the complexities and issues arising from a population of individuals with disparate interests attempting to combine the views of its individuals in a structured manner. The objective of such a process is usually to determine a collective policy for the population's future behavior. Such a process is core to the notions of democracy and self-determination. If a population can be said to have self-determination, it must reliably be able to determine its own collective course of action. Politics is the name given to this process when the population is humans. The broader term social choice refers to any society. Recent research in artificial intelligence gives rise to concerns of social choice applied to populations of artificial intelligence agents [ER91, ER93]. Of special concern in this setting is the availability of computational resources to individuals in the population. Considering the availability of computational resources to members of the society, methods and protocols must consider both the positive and adverse effects such computational power could have on the process. This gives rise to the relatively young field of computational social choice, which examines precisely these consequences.

A general method for combining the views of many individuals is by *voting*. When voting, individuals record their opinions about the various alternatives to some medium, and the votes are pooled. A predetermined rule is then applied to the pool of votes to determine the winner.

Informally, a *voting rule* is a function that selects in a prescribed manner a winner from a group of candidates using a collection of votes. These votes often express more than simply a preference for one alternative over all others—the most common case in human political elections. It is possible for voters to express approval and disapproval of alternatives, to rank the alternatives, and even to enumerate the strength of their preference for the alternatives. Most voting rules examined in the literature use the definition of a vote as a complete ranking of all the alternatives, without ties. We use this model here.

When comparing voting rules, social choice researchers examine them on the basis of their satisfaction of certain criteria: intuitive properties which are seen as good properties for voting rules to have. There are intuitive behaviors voting rules should exhibit. The central idea behind voting is that the voters collectively choose the outcome with their votes. Therefore the voting rules should respond to the voters' votes. A voting rule is called Pareto-optimal if when every voter prefers some alternative to some other alternative the rule should not select as the winner the alternative unanimously disliked by the voters. Being Pareto-optimal does not require that a rule always select a candidate that is preferred by a majority of the voters—there may not be such a candidate—but intuitively a voting rule should never choose a candidate if there is another candidate preferred by everyone.

Another criterion that seems sensible for a voting rule is that it use information from more than one ballot. If a rule only looked at one ballot, and selected that ballot's top ranked choice as the winner, it would be Pareto-optimal, but we would hope that a rule considers the input of all the voters. If a rule only considers one ballot, we say that the rule is dictatorial.

These two criteria are derived from a basic notion of fairness. It is hard to argue that a voting rule need not satisfy the above criteria. Consequently, every voting rule that is seriously considered for actual use satisfies these basic requirements.

Two other important criteria to consider when examining voting rules are monotonicity and independence-of-irrelevant-alternatives (abbreviated

IIA). These criteria are in a sense stronger than the Pareto-optimality and non-dictatorship. There are many sensible voting rules that do not satisfy these criteria, including many voting rules used in practice.

Monotonicity in a voting rule considers how the outcome of an election would change by moving a candidate up on some of the ballots. If moving the candidate up on some ballots (and down on none) leads to that candidate doing worse in the overall election, that would violate monotonicity. This need not be a common occurrence: if it is at all possible for such a situation to arise, then the voting rule is non-monotonic. Non-monotonicity means moving the current winner up on some ballots could cause that candidate subsequently to lose the election. In one sense, monotonicity is very important to voters having confidence in the electoral process. A voter should feel that a vote for a particular candidate helps that candidate win. If a voting rule is not monotonic, the voter cannot be sure of that. The voter may wonder what is the point of voting if expressing their support for a preferred candidate may cause that preferred candidate to do worse. A non-monotonic voting rule is not completely untrustworthy however: A non-monotonic voting rule may in fact only fail the monotonicity criterion for a few cases. An example of a non-monotonic voting rule is the instant runoff voting rule. Consider the following election between three candidates a, b, and c. Suppose 39 voters vote $a > b > c$, 35 vote $b > c > a$, and 26 vote $c > a > b$. The victor here is a: No candidate has a majority, so c is eliminated first. The 26 votes of $c > a > b$ transfer to a, making a the winner. However, if 10 of the $b > c > a$ voters raise a to the first position on their ballots (becoming $a > b > c$ voters), c is now the winner: b is the first eliminated, and the 25 remaining $b > c > a$ votes transfer to c, giving a majority and a victory to c. Thus the instant runoff voting rule is non-monotonic.

IIA states that irrelevant candidates do not alter the outcome of the election. Declaring a candidate irrelevant in a political election is likely to raise ire from that candidate's supporters, but as far as social choice research is concerned, an irrelevant candidate is any candidate that does not win. A voting rule that is unaffected by irrelevant candidates produces the same

winner if any or all of the non-winners did not run, given that the order of the other candidates on each ballot remains the same. For example, in an election under the plurality voting rule with three candidates, a, b, and c, where a and b each have 41% and 39% of the vote respectively, c has 20% of the vote, and c's supporters prefer b to a (after c)—a is the winner of the election as stated, but if the loser c were to drop out, b would become the winner. Thus plurality does not satisfy IIA.

One of social choice's earliest and most well-known results is Kenneth Arrow's impossibility theorem, which colloquially states that there is no perfect voting rule [Arr51]. Arrow's impossibility theorem states that there is no Pareto-optimal, non-dictatorial voting rule that operates on more than two candidates which is also monotonic, and for which irrelevant candidates never alter the outcome of the vote for the other candidates.

Monotonicity, Pareto-optimality, non-dictatorialism, and independence-from-irrelevant-alternatives are all properties examined when comparing voting rules. Arrow's Theorem shows that they cannot all be simultaneously satisfied by a single voting rule if the election has more than two candidates. Pareto-optimality and non-dictatorialism are relatively easy to satisfy and it is hard to argue that a voting rule that does not satisfy them is reasonable. Voting rules that are used in practice typically violate monotonicity or, more often, IIA. As shown in the examples above, plurality violates IIA, and instant runoff voting violates both monotonicity and IIA. The Borda count and the veto rule violate IIA.

Two other disappointing results for the concept of fairness in social choice are the Gibbard-Satterthwaite and Duggan-Schwartz theorems. Generally speaking, we would like to think of voting as a way to combine people's honest preferences. In this regard it would be undesirable for some voter to be able to record preferences on the ballot that are different from its true preferences, and in so doing achieve a better result for itself at the expense of the general social welfare (i.e., causing the outcome of the election to be different than what it would have been if all voters voted their true preferences). Such an attack is called *manipulation* in the social choice literature. The Gibbard-Satterthwaite theorem holds that every non-dictatorial vot-

ing rule that picks a single winner and does not preclude any candidates from winning admits a situation for which some voter achieves a better result for itself by misrepresenting its preferences [Gib73, Sat75]. Duggan and Schwartz generalized Gibbard-Satterthwaite for voting rules that select multiple winners. The Duggan-Schwartz theorem added that as long as the rule does not simply declare every election a tie, then that rule also admits a situation for which some voter achieves a better result by misrepresenting its preferences [DS00]. However, neither of these results imply that a voter intending to manipulate the election by voting disingenuously can always do so, or moreover, even if they can, that they can find such a manipulation easily.

A malicious artificial intelligence agent or group of agents could easily have direct access to computational resources to aid in selecting a vote or votes that will produce the best outcome for the malicious group, which may be very different from the preferred outcome for the whole voting population. Moreover, it is conceivable for the attackers to know how the other members of the group will vote. A careless voting protocol design may allow for eavesdropping on votes as they are collected through a man-in-the-middle attack. In the case of voting by artificial intelligence agents it is possible that the malicious agents could have copies of the source code of the other agents. In this case they could simply run instances of the non-malicious agents using the their own private computational resources to determine the votes of the others.

Bartholdi, Tovey and Trick first examined the computational hardness of finding a manipulation beneficial to the manipulator [BTT89a]. Suppose a voting rule we wish to use has situations where a voter can achieve a better result for itself through manipulation. If that voter is unable to efficiently find such a manipulation, that will likely deter strategic voting. Bartholdi and Orlin showed that finding a beneficial manipulation under the single transferable vote rule is NP-complete [BO91], even with one manipulator. Their result however requires that the number of candidates and the number of voters be unbounded. If either are bounded, and the votes are unweighted, then the problem is in P by brute force (for voting rules that are in P).

Conitzer and Sandholm began to examine the complexity of both constructive and destructive manipulation when the number of candidates is fixed in [CS02]. Constructive manipulation asks if the coalition can make a specific candidate become a winner, while destructive manipulation asks if the coalition can merely *prevent* a specific candidate from being a winner. They determine the complexity for constructive and destructive manipulation under several voting rules for which determining the winner is in P, for cases where the manipulator votes are weighted, as well as for cases where the manipulators do not have perfect information about the non-manipulators' votes but have only probability distributions for their votes. They show that with a fixed number of candidates and a voting rule whose winner problem is in P, unweighted constructive coalitional manipulation is in P, as there are only polynomially many ways for the coalition to vote.

Conitzer and Sandholm also found a method of modifying any voting rule to raise its difficulty to NP-hard by adding a pre-round which eliminates half of the candidates by simulating the first round of a binary single-elimination tournament [CS03]. The candidates are paired according to some schedule, and the pairwise election is held between each pair. The losers of these preliminary contests are immediately eliminated, and the winners go on to participate in the election under the original voting rule. Depending on the method used to generate the pairings for this first elimination round and when the pairings are published in comparison to when the votes are cast, the technique elevates the complexity of manipulation to NP-hard, #P-hard, or PSPACE-hard.

Elkind and Lipmaa adapted the technique of [CS03] to secure the pre-round against influence by election administrators by generating the pairings with one-way functions (assuming one-way functions exist), and in so doing, expanded the hardness guarantees to more than one manipulator [EL05].

While it is interesting to know that there are universal tweaks that can be made to a voting rule to raise the difficulty of its manipulation, the modifications they describe are hard to argue for. They alter voting rules in such a way that the modified rules seem fundamentally different. Supporters of candidates eliminated in the pre-round will argue that their candidate

was prematurely eliminated. It is an aphorism that the point of voting is to convince the losers that they lost. If the goal of voting is indeed to convince the losers and their supporters that they lost, then these pre-rounds may prevent this. In a close race, the pre-round schedule may be the sole differentiation between a winner and a loser, and a different schedule may yield a different outcome. This is undesirable in general.

Additionally, this line of research considers only worst-case hardness measures like NP-hardness. It may be the case that for some elections, under a given voting rule, a manipulative voter or coalition of voters may not be able to influence the result at all. In some other elections, they may be able to alter the outcome of the election only after great difficulty and much computation. In still other elections, they may be able to alter the outcome the election quite easily. While establishing NP-hardness for a manipulation problem guarantees no efficient algorithm for all cases (without admitting P = NP), we would generally like to be sure that a coalition of manipulators cannot efficiently find a beneficial manipulation in a typical election. Perhaps a great fraction of the elections that are likely to arise in practice are easily manipulable. The hardness model that captures this idea is that of a problem being "usually hard." This hardness model is seen in the study of cryptography. It is not acceptable for a cryptographic system to be merely NP-hard, or even hard "on average" in the Levin sense [Lev84], since it is still possible for there to be only $1/poly(n)$ of the instances which are hard. A cryptographic system must be hard in the vast majority of individual instances that are likely to arise in practice, preferably having zero, or only $O(1)$ weak instances like, e.g., the weak keys of DES [TW01]. Moreover, it is not entirely clear which distribution of votes is the most appropriate to consider. Black and many other authors have argued that uniform distribution is in fact not representative of electoral preferences in many cases, and that the single-peaked distribution is more appropriate [Bla48, Bla58, NW87, DHO70, PR97, Kre98].

Conitzer and Sandholm first raised the question of manipulation complexity in the average case in [CS03], and Elkind and Lipmaa showed that it is not possible to elevate the average case complexity of manipulation of

a voting rule in P simply by adding a pre-round to it [EL05].

Conitzer and Sandholm continued to explore of the possibility of average case hardness in [CS06]. They showed that for any monotonic voting rule which is executable in polynomial time, when there are exactly two candidates that could ever possibly be made to win considering the non-manipulators' votes and the entire combinatorial space of all possible ways the manipulators could vote, there is a polynomial-time algorithm which finds both of these candidates and provides a corresponding set of manipulator votes to make each the winner. The basic idea of the algorithm is as follows. If the voting rule is executed on the non-manipulators' votes alone, then the winning candidate there is one of the possible winners, because the manipulators can cast ballots with that same candidate in the first position, and under a monotonic voting rule, that candidate will remain the winner. Finding the second possible winner involves looping over all other candidates to test their electability by computing the result of the voting rule on the union of the non-manipulators' votes and the manipulators' votes, placing the currently winning candidate in last place on all of the manipulators' ballots, placing the candidate being tested in first place, and using an arbitrary ordering for the rest of the candidates. Since, by assumption, the manipulators could make exactly two candidates win, the ordering of the other candidates does not matter and the given votes are sufficient to make the other candidate win. They go on to demonstrate that under a specific distribution of non-manipulator votes and for a fixed size coalition, the frequency of cases where the manipulators can make three or more candidates win is small when compared to the frequency of cases where exactly two candidates can be made to win. From this they conclude that average case hardness is not possible for monotonic voting rules.

Their algorithm is unable determine however if, in fact, there are exactly two possible winners. If there are more than two possible winners, it could produce one, two, or any number up to all of these possible winners.[1] Furthermore, it gives no concrete determination of the actual number of possible

[1]The algorithm as presented in their paper stops after finding the second possible winner, but it is easily modifiable to continue the search for more.

winners, nor a resolute answer on the electability (under manipulation) of a particular candidate if it did not give a witnessing set of manipulator votes.

They remark that instances in which only one candidate could ever be made to win are uninteresting, since the manipulators are out of luck and are unable to alter the outcome of the election. In the case where exactly two candidates can win, the manipulators' job is easy, since the polynomial-time algorithm could be applied to find both winners and the precise votes needed to make them win. The manipulators' job then can only be hard in instances where more than two candidates could be made to win. But it is not easy to determine, generally, when this algorithm reports that some of the candidates can win, if any other candidates could have won under an appropriate manipulation. The usefulness of this algorithm depends on how often in practice it is possible for a manipulating coalition of a given size to make one, two, or more of the candidates win. The distinction between one candidate being the only possible winner, and the algorithm reporting only one possible winner is significant. If a coalition can make three or more candidates win, but is only able to make a candidate other than the current leader win by voting differently from one another, Conitzer and Sandholm's algorithm would report only one possible winner, and miss the other two.

The question then is how often do these cases that could cause a problem for Conitzer and Sandholm's algorithm arise in practice? Further empirical investigations into the frequency at which these instances occur would provide some insight into the applicability of their result. How often can exactly one candidate be made to win, when two, and when more?

The distributions they examined tend toward increasing distinction between successive candidates as the number of non-manipulators increases. Consequently, this distribution underrepresents instances where there are comparably-sized camps of like-minded voters with strong conflicts in the preference orders between the camps. These distributions would seem to be the most promising place to find hard instances. Additionally, they only examined fixed-size manipulating coalitions. They did not vary the coalition size with the number of non-manipulators. As the number of non-manipulators grows, the fixed-size coalition is unsurprisingly seen to have

rapidly diminishing influence over the election.

What happens when the number of manipulators is varied in proportion to the number of non-manipulators? If the non-manipulators' votes are being drawn consistently from a known distribution, it is intuitive that proportionally increasing the number of manipulators would yield similar amounts of influence to the manipulators. Conitzer and Sandholm do note that if three or more candidates can be made to win, it is possible that some of those candidates can only be made to win if some of the manipulators vote differently than others. Having to vote differently from one another is not a concern for the manipulators when only two candidates can be made to win. When three or more can be made to win, this does become important, especially considering that if the relative fraction of manipulators to non-manipulators is large and the manipulators have more power, it becomes important not to make a candidate they dislike become the winner accidentally by, say, placing that candidate in the second position on all of their ballots.

Procaccia and Rosenschein revealed a clearer picture of the region where manipulation problems are interesting and possibly difficult [PR07]. Coalitions that are too small will often not have enough power to change the result of an election. Coalitions that are large will often be able to do so easily. Procaccia and Rosenschein put bounds on the fraction of votes that can come from manipulators for the problem to be hard with any frequency. They demonstrated based on the Central Limit Theorem and Chernoff's bounds that as long as the distribution of votes are minimally random and that the individual votes are independent and identically distributed, then only the manipulation problems where the total weight of the manipulators is $\Theta(\sqrt{n})$ could be hard, where n is the number of non-manipulators. If the size of the coalition is $o(\sqrt{n})$, it is too small — the probability that they will be able to influence the outcome tends to 0 as n increases. If the coalition size is $\omega(\sqrt{n})$, it is large enough to easily change the outcome — the probability that they can manipulate tends to 1 as n increases.

The above result would seem to suggest a critical size for the manipulating coalition in order for manipulation problems to be hard, and that

critical size appears to be around $m = \Theta(\sqrt{n})$. Walsh investigated empirically the difficulty of manipulation when the coalition was near this critical size [Wal09b]. For a coalition of manipulators with weighted votes, with uniformly randomly distributed weights, and total weight equal to $c\sqrt{n}$ (c is the constant implied by Θ-notation), Walsh observed a smooth transition in the probability of a random election being manipulable under the veto voting rule. Worthy of remark is that the probability of instances being manipulable smoothly changed from 0 to 1 as the function $1 - \frac{2}{3}e^c$. The shape of the transition did not change as n was increased. The smooth transition resembled more the transition of problems with known polynomial algorithms than it did the transition of NP-hard problems, whose transitions resemble a sigmoid shape and grow more to resemble a step function as n increases.

Walsh also investigated the manipulability of single transferable vote by a single manipulator, as well as the difficulty of the search to find a beneficial manipulation or prove that none exists [Wal09a]. The analysis considered elections with up to 128 voters and 128 candidates. Votes were drawn from the uniform distribution, the Polya Eggenberger urn model [Ber85] (a distribution over elections that tends toward highly correlated votes), as well as two sampled real world voting data sets. Using an improved form of a method given by Conitzer [Con06], Walsh was able to determine whether elections could be manipulated by a single manipulator at a very low computational cost, even with 128 non-manipulators and 128 candidates to order. While the theoretical upper bound on the computational complexity of Conitzer's algorithm was $O(n1.62^m)$, the observed behavior of the algorithm was closer to $O(n1.008^m)$ for uniform votes, and $O(n1.001^m)$ for elections drawn according to the urn model. In practice this meant it was easy to compute the manipulability status of the sample elections.

Friedgut, Kalai and Nisan made a troubling discovery about the average case difficulty of manipulation [FKN08]. This result has been referred to as a "quantitative version" of Gibbard-Satterthwaite. Their result applies to voting rules that are neutral and far from being a dictatorship. A voting rule is neutral if it is blind to the identities of the candidates (i.e., the result of the election commutes with all permutations of the candidate set). The distance

from a dictatorship is measured by how often the result of the election differs from the will of any single voter, i.e., a potential dictator. Consider that it is possible to design a voting rule that functions as a dictatorship in all cases except, say, for one very precise profile of votes. Such a voting rule would be considered not a dictatorship. The notion of distance from a dictatorship formalizes the need for the voting rule to consider the input of at least two and hopefully many voters, and that it should do this most if not all of the time. This condition is less strict than requiring the rule to be anonymous, i.e., requiring that it treat all voters identically, but certainly an anonymous rule would satisfy it. Friedgut et al. showed that as long a voting rule is neutral and far from a dictatorship, then it will be manipulable in the average case by a single manipulator, under the uniform distribution of votes, for elections of exactly three candidates. Though they were only able to prove their theorem for the case of exactly three candidates, most of their proof generalizes to more than three candidates, and they conjecture that their theorem holds in the general case.

Xia and Conitzer proved a theorem similar to the one proved by Friedgut et al. which does extend to the general case of more than three candidates [XC08]. While their theorem does place more requirements on the voting rule in question, nevertheless many popular voting rules still satisfy the conditions. In particular, scoring protocols, single transferable vote, Copeland (with 5 or more candidates), maximin, and ranked pairs all meet the conditions.

Isaksson, Kindler and Mossel were able to fully extend Friedgut et al.'s theorem to the general case, in both number of voters and number of candidates [IKM10]. They were further able to give an exact formula for the probability of manipulation, using the same assumptions about the voting rule and distribution of non-manipulator votes as Friedgut et al. Under the uniform distribution of votes, a coalition of voters can find a manipulation with probability at least $10^{-4}\epsilon^2 n^{-3} m^{-30}$, with ϵ being the distance of the voting rule from a dictatorship, n the number of non-manipulators, and m the number of candidates.

Ariel Procaccia explored the idea of circumventing these results by con-

sidering randomized voting rules [Pro10]. The idea here is to use a voting rule that selects a winner randomly, where this randomly selected winner will always have a score that is within a γ-fraction of the maximum score. Such a voting rule is said to approximate the original voting rule with an approximation ratio of γ. This method could be applied to any voting rule that has a definable notion of score, and is not restricted merely to scoring protocols. The argument for using such a voting rule is that the approximation will usually select an alternative that is optimal or nearly optimal, and this approximation will be invulnerable to strategic voting.

It is important to note that the work Friedgut et al., Xia and Conitzer, and Isaksson et al. are all based on the uniform distribution. Faliszewski, Hemaspaandra, Hemaspaandra, and Rothe showed that many manipulation problems known to be NP-hard in general fall down to P under a single-peaked preference distribution [FHHR09].

1.1 Overview

The aim of this work is to further explore the practical difficulty of the manipulation problem. The empirical analysis will explore the manipulability of elections over the uniform distribution, and two instances of the spatial and Condorcet family of distributions.

Under the uniform distribution, all rankings of the candidates are equally likely. We examine the uniform distribution because it is a natural an impartial distribution of votes, and it is commonly considered in other research into manipulation as well.

The Condorcet family of distributions generate ballots that are biased to put the candidates in one specific order, with a parameterized amount of bias. The Condorcet distribution with $p = 0.6$ is examined for comparison with the results in [CS06].

In the family of spatial distributions, every voter and every candidate has a position on n orthogonal one-dimensional "issues." Any particular voter ranks the candidates by their euclidean distance from the voter in the n-dimensional space formed by the issues, the nearest candidate being the most

preferred. The spatial distribution is examined for several reasons. It seems to model the perceived political landscape in many human populations, and it is used in other empirical analyses of properties of voting rules and the frequency at which they exhibit those properties [Cha85, CC78]. The spatial distribution with a single dimension is actually the single-peaked model mentioned earlier.

This work examines elections with three and five candidates. We examine elections with number of non-manipulators equal to a power of two, from 1 to 128. We consider the manipulability of elections using the voting rules plurality, plurality with two-candidate runoff, instant runoff voting (a.k.a. single transferable vote, alternative vote, and Hare voting), the Borda count, and veto.

The analysis uses a randomly generated election dataset. The implementation of the dataset generator, as well as the dataset itself and all code written by this author for the analysis can be found in Appendix A.

Chapter 2

Preliminaries

We will now provide definitions for the some of the terms and mathematical structures to be used in the rest of this report, as well as background on other topics addressed.

2.1 Elections

An *election* E is a set of candidates paired with a list of votes (C, V). C is the *candidate set* of the election. The number of candidates, $|C|$, is also called m. When referring to specific candidates we will use letters a, b, and c.

When not referring to specific candidates but rather candidates in general, we'll start at c and use letters c, d, and e. V is a list of votes cast by voters. In this work we assume that each voter casts a single vote, and that all the votes have equal weight. Thus the number of votes in the election's vote list is identical to the number of voters in the election, and is sometimes written as n. We have not yet introduced the possibility of manipulation, so none of these voters are part of any manipulating coalition. They cast their votes according to their honest preferences of the candidates.

The voters are numbered from 1 to n. Because the voters do not take any action beyond casting their vote, we may identify voters with their votes. Each vote v_i, the vote cast by voter i for $1 \leq i \leq n$, is a total ordering over

the candidates. Truncated ballots, i.e., ballots where the ranking stops after listing some subset of the candidates, are not allowed, nor are ties in the ranking, nor are expressions of relative strength of preference for candidates. When referring to the relative position of candidates for a particular vote, we may use the notation $a >_i b$ to indicate that a is ranked higher than b on ballot v_i.

An election is evaluated with a *voting rule*, called R, which produces a subset of the candidates as a set of winners. Mathematically speaking, it is a total function from a candidate set and a list of votes on that candidate set to a subset of the candidate set. The candidates in this subset are considered the winners of the election. If the winner set contains just a single candidate, that candidate is the unique winner of the election. If the winner set contains more than one candidate, they are all considered non-unique winners of the election. In this work, we are interested only in unique winners. Therefore, we consider any result set containing more than one candidate to be the same as the empty set. If we discuss manipulators making a candidate a winner, or say a candidate can be made to win, we mean that the manipulators can make that candidate the unique winner in at least one case. The set of *possible winners* is the set of all candidates such that each one can separately be the unique winner under an appropriate set of manipulator votes. An example can be found in Section 2.4.

In an election, the voting rule R is applied to C and V, and $R(C, V)$ gives the winner set. It is generally desirable that the function R be computable in polynomial time in the sizes of C and V. If R is not computable in polynomial time, the winner of an election may not be determinable in practice. While there has been some study of voting rules which are NP-hard to compute and exploration of how often they are solvable [BTT89b, HH06], in practical elections the voting rule must be polynomial-time computable. Thus it is the case for all voting rules that are considered usable in practice that they are computable in polynomial time, and hence we consider only polynomial-time computable rules in this work.

2.2 Voting Rules

This section gives definitions of the voting rules which are used in this report, with related material.

Definition 2.1 (Pairwise Election)**.** Given an election E and two candidates a and b in the candidate set of E, a pairwise election between them considers only the relative order of a and b on the votes in the vote set V, treating all other candidates as if they were not part of the election. The candidate that is higher on more ballots is the winner. A pairwise election may be a tie. Voting rules that use pairwise elections in the process of determining a winner will specify what occurs in the event of a tie.

Definition 2.2 (Voting Rules)**.** Below are the definitions of the six voting rules we analyze in this report. All of these rules as stated below have the possibility of being tied. Since we are concerned only with unique winners in this report, we do not consider any candidate a winner if there is a tie or multiple winners.

- Under the Plurality rule, each candidate gets one point for each vote putting that candidate in the first position. The candidate with the greatest score is the winner.

- Instant runoff voting, a.k.a. single transferable vote, alternative vote, and Hare voting, selects the winner through a series of rounds. In each round, the candidate in the first position on the fewest number of ballots is eliminated from further consideration. If more than one candidate is tied for this dishonor, all such candidates are eliminated. The remaining rounds proceed as if any eliminated candidates were not in the candidate set nor in any of the votes. If any round eliminates all remaining candidates, all the candidates remaining going into that final round tie for the victory. If no ties occur, exactly one candidate will remain after $m - 1$ rounds.

- Plurality with two-candidate runoff, a.k.a. plurality-with-runoff, operates in two rounds: the first round eliminates all but the two candi-

dates who have the greatest number of first place votes, then applies plurality to those remaining candidates to select the winner.

If more than two candidates are tied for most first place votes in the first round, those candidates all advance. If two or more candidates are tied for second-most first place votes in the first round, those candidates are all eliminated.

- The Borda count awards 0 points to the candidate in the lowest position on the ballot, 1 point to the next higher candidate, and so on, with the candidate in the first position receiving $m - 1$ points. The candidate with the highest sum score over all the votes is the winner.

- The Veto rule awards 1 point to all candidates not in last place on the ballot. The candidate with the highest sum score over all the votes is the winner. Alternately, each voter *vetoes* a single candidate, and the candidate with the fewest vetoes is the winner.

- The winner under Copeland's rule is the candidate with the most points. Points are awarded based on pairwise elections. A win in a pairwise election earns a candidate 1 point, a loss 0. The amount awarded for a tie is parameterized. The most common is $\frac{1}{2}$ point, with 0 and 1 also being common alternatives, but any rational value $0 \leq \alpha \leq 1$ is allowed. Faliszewski, Hemaspaandra, and Schnoor show that for all values of α except $\frac{1}{2}$ manipulation is NP-hard, even if the number of manipulators is limited to 2 [FHS08, FHS10]. The difficulty of manipulation for $\alpha = \frac{1}{2}$ is still unknown. In this work we used $\alpha = \frac{1}{2}$, as it is the most common value seen in practice.

There is a generalization of some voting rules into a family called *scoring protocols*. A rule in the family of scoring protocols is called a scoring protocol.

Definition 2.3 (Scoring Protocol)**.** A scoring protocol has a vector of point values α of length equal to the size of the candidate set, with $\alpha_i \geq \alpha_{i+1}$ for all i. For each vote, each candidate receives points according their position on

the ballot: the candidate in the ith position receives α_i points ($1 \leq i \leq n$). The candidate with the most points wins.

Plurality is a family of scoring protocols with scoring vector $\langle 1, 0, \ldots, 0 \rangle$. Veto is a family of scoring protocols with $\langle 1, 1, \ldots, 1, 0 \rangle$. The Borda count is a family of scoring protocols with $\langle |C| - 1, |C| - 2, \ldots, 0 \rangle$.

2.3 Election Distributions

The distribution from which instances are drawn is critical to an empirical analysis.

Definition 2.4 (Probability Distribution)**.** A probability distribution is a function from a set of possible mutually-exclusive outcomes to a real-valued probability of each outcome's occurrence.

Alternatively, a probability distribution can be specified by a generation function that makes use of a randomness-source. When such a generation function is called it produces a representation of an outcome. Each outcome is produced with probability according to the distribution function.

Here we specify the three distributions over election instances we will be using to sample the election space by specifying their generation functions. Each generation function used in this work takes a size parameters n and m and generates an election from the set of all elections with n voters and m candidates. Each of the distributions we examine draws each of the n votes independently and distributed identically, and thus it suffices to specify an election distribution by a generation function for a single vote.

The uniform distribution, also known as the impartial culture distribution, is the simplest distribution. Each voter's preferences are selected uniformly from all possible permutations of the candidates.

The spatial distribution attempts to model the political landscape that is thought to be seen in many human populations. A voter can be thought of as a vector in a d-dimensional vector space, each candidate is also seen as a vector in d-dimensional space, and the preferences of a voter are determined

by sorting the candidates by their euclidean distance to the voter, nearest-to-farthest. Each of the d dimensions is an *issue* in the election. Issues are treated as orthogonal to one another. Furthermore, each coordinate in the positions of each voter and each candidate are distributed according to a normal distribution with mean 0 and standard deviation 1.

The Condorcet distribution imagines that there is an intrinsic "true order" for the candidates, and higher candidates are simply better than lower candidates. Each vote drawn from this distribution is drawn using the following naïve method: for each pair of candidates where $a > b$ (a is intrinsically better than b), candidate a is ranked above candidate b on the ballot with probability p, a parameter; with probability $(1 - p)$, b is ranked higher than a. If at any time in this process a cycle is introduced, the vote is discarded and the process starts over. Generating votes in this manner is a time-consuming process because restarts occur frequently, especially with more candidates and values of p closer to 0.5. Nevertheless, the naïve method was not prohibitively time-consuming for generating our data set. The parameter p can be thought of as a level of noise each voter must contend with in receiving the "true order." Setting p to values $0.5 < p < 1$ will lead to the election being from slightly to extremely lop-sided. A value of $p = 0.5$ degenerates to the uniform distribution.

2.4 Manipulation

All voters in the set V are non-manipulators. In manipulation, a coalition of voters, called manipulators, coordinate in selecting the votes they cast, in an attempt to get an outcome they prefer more than the natural outcome that would have occurred if they had all voted their true preferences. We may refer to the size of the manipulating coalition as k. The manipulators cast a set of k votes. The set of their votes is denoted M.

For example, suppose an election has three candidates a, b, and c. The non-manipulator votes V are 10 votes with $a > b > c$, and 8 votes with $b > a > c$. Suppose the plurality rule is being used.

If there are $k = 3$ manipulators and their true preferences are $c > b > a$,

and they vote honestly, their least preferred candidate, a, will win. They would do better for themselves by voting $b > c > a$. Then b, a candidate they prefer to a would win. For this non-manipulator vote set V and $k = 3$, the set of possible winners is $\{a, b\}$. a is in the set because regardless of the manipulators' true preferences, a could be a unique winner for some set of manipulator votes. If k were only 2, the manipulators could only get b to tie with a. Since the set of possible winners contains only those candidates the manipulators could make unique winners, the possible winner set for $k = 2$ is just $\{a\}$.

For manipulation in this work, the manipulators are a completely separate and distinguished group from the other voters in the election. If the voters in the election number n, and the manipulators number k, there will be a total of $n + k$ votes passed into the voting rule. The set of votes passed to the voting rule is $V \cup M$. We do not give the manipulators an explicit set of true preferences. The true preferences mentioned in the above example were for illustrative purposes. We are concerned only with the candidates in the possible winner set. We assume that if the manipulators can calculate the possible winner set, they can decide amongst themselves which candidate is the best choice—in the above example c could not be made a unique winner, and b is their best option. It is interesting to observe that if different members of the manipulating coalition have different true preferences over the candidates, they may have to resort to an election amongst themselves to decide who to manipulate for. Nevertheless, we do not concern ourselves specifically with true preferences any further in this work.

The manipulators are assumed to have perfect knowledge of the non-manipulators' votes, and may use that knowledge to decide how to cast their own votes. This is a commonly made assumption when investigating the hardness of manipulation. Assuming perfect knowledge is the most favorable conceivable circumstance for the manipulators. Any results demonstrated under this assumption must surely hold in less favorable conditions. Conitzer and Sandholm use this model for Constructive-Coalitional-Weighted-Manipulation in [CS02], for Constructive-Manipulation in [CS03], and exclusively in [CS06]. We only consider constructive manip-

ulation in this work. We do not consider destructive manipulation.

Other definitions are possible, like the one used by Chamberlin [Cha85]. In Chamberlin's work, the manipulators are not a separate group, but instead they are the subset of voters whose true preferences do not agree with the result of the election computed using all the true preferences. These are the voters who would be motivated to work for a different outcome. In [EL05], Elkind and Lipmaa consider the manipulator (they only look at coalitions of one) to be a member of the voting population (equivalent to V here), and not in a separate group. Under this formulation, the manipulation problem is similar to the problem of bribery. In bribery, it is asked if one can select a group of voters from the existing voting population to bribe into voting as the briber chooses, and thus swing the result of the election. A fixed budget is given as part of the input, and the briber must select the voters to bribe as well as their new voters. Bribery has forms that allow for weighted voters, and voters with different bribery prices. Bribery of an election using the plurality voting rule is in P, even if voters are given weights or prices, but notably it becomes NP-complete if voters have weights *and* prices.

We now define manipulation instances as we use them in this work. Manipulation instances we examine come in several forms. All forms will have a candidate set C, a set of non-manipulator votes V and a voting rule R. There may also be a preferred candidate c_p, an integer k, the size of the manipulating coalition, or both. The votes that the manipulators eventually cast is denoted as M. Since the votes are to be decided by the manipulating coalition, M is not part of the input of any manipulation instances, but rather refers to the votes the coalition eventually decides to cast.

Much of the literature does not consider the voting rule R to be part of the input in a manipulation instance. Rather, manipulation is seen as a family of problems, and manipulation for each rule is discussed separately (e.g. Borda-manipulation, Copeland-manipulation, etc.). We have chosen in this work to view R as part of the input, because the method we use to solve manipulation instances is agnostic to which rule is actually used.

A manipulation instance with both c_p and k, i.e., (C, V, R, c_p, k), is a

decision problem. Solving such an instance entails deciding whether or not k manipulators can make c_p a unique winner when voting rule R is applied to $V \cup M$. An algorithm for solving such an instance may or may not give a witnessing manipulator vote set.

If a manipulation instance does not give a preferred candidate, i.e., a manipulation instance (C, V, R, k), the task is to determine which candidates in the candidate set can be made unique winners by k manipulators. In other words, to compute the possible winner set. An algorithm for solving this form of manipulation instance will return the subset of the candidates $W \subseteq C$ such that for each $c \in W$ there exists a set of k manipulator votes that makes c a unique winner. The algorithm may or may not give a witnessing set of manipulator votes for each such candidate. Additionally, the absence of a candidate c from W indicates that k manipulators cannot make c become a unique winner. Examples of such algorithms include Greedy-Manipulation from [BTT89a], and Find-Two-Winners from [CS06].

If a manipulation instance does not give a coalition size but does give a preferred candidate, i.e., a manipulation instance (C, V, R, c_p), the task is to compute the minimum number of manipulators required make c_p a unique winner. Algorithms solving such instances will compute this minimum value. We call this value the *manipulation number* for the candidate c_p with respect to the manipulation instance (C, V, R), or equivalently, the minimum coalition size needed to make c_p a unique winner.

If a manipulation instance specifies neither c_p nor k, then the task is to compute the manipulation numbers for each candidate $c \in C$. Recall from before one of the primary questions we are evaluating is how often can only one candidate be made to win, when exactly two, and when more. Under reasonable assumptions about the voting rule which we will make precise later, that question is asking for the manipulation numbers of the candidates. The possible winner set of a manipulation instance (C, V, R, k) contains exactly those candidates whose manipulation numbers with respect to (C, V, R) are less than or equal to k. Moreover, we are not concerned with the identities of the candidates, only the size of the possible winner set for a given value of k. Thus, if we ignore the identities of the candidates, we

can define the concept of *manipulation numbers* for a manipulation instance (C, V, R).

Definition 2.5 (Manipulation numbers)**.** The manipulation number of a candidate c with respect to a manipulation instance (C, V, R) is the minimum number of manipulators that are required to make c a unique winner of (C, V, R).

The manipulation numbers for a manipulation instance (C, V, R), are a sequence of m non-negative integers. The jth manipulation number ($1 \leq j \leq m$), is the minimum number of manipulators required to make the possible winner set contain at least j candidates.

The jth manipulation number may be denoted as M_j.

Alternatively, if the jth manipulation number of the manipulation instance (C, V, R) is k, then with k manipulators, at least j of the individual candidates' manipulation numbers with respect to (C, V, R) will be less than or equal to k.

It what follows, when we refer to the manipulation number of a candidate, it will be assumed to be with respect to a manipulation instance (C, V, R). Otherwise we mean the sequence of manipulation numbers for the instance itself.

By examining manipulation instances of the form (C, V, R) in terms their manipulation numbers, we can show some useful properties. As we have defined manipulation numbers for candidates, there is currently no guarantee that if k manipulators can make a candidate a unique winner, that $k + 1$ can as well.

Thus it would be helpful if our voting rules had the property mono-add-top, defined by Woodall [Woo94]. It is a weaker form of the participation criterion, i.e., participation implies mono-add-top. Mono-add-top can be considered a relative of monotonicity as used by Arrow [Arr51] (Woodall refers to that property as mono-raise). We reproduce a definition of mono-add-top here.

Definition 2.6 (Mono-add-top)**.** In an election (C, V) with voting rule R, R satisfies mono-add-top if whenever a candidate c is the unique winner

of the election and additional ballots are added with c at the top and are otherwise arbitrary, c remains the unique winner.

Strictly speaking, we do not require the full power of mono-add-top. Mono-add-top requires *all* ballots with c at the top allow c to remain the winner. All we need is a guarantee that there is *some* ballot that allows c to remain the winner. It is quite conceivable that even with a voting rule that does not satisfy mono-add-top, that the extra manipulators beyond the k required could find a way to vote that preserves the unique winner. This existential version of mono-add-top is clearly implied by regular mono-add-top.

Definition 2.7 (Existential-mono-add)**.** In an election (C, V) with voting rule R, R satisfies existential-mono-add if whenever a candidate c is the unique winner of the election then there exists some way to cast an additional ballot so that c remains the unique winner.

Using voting rules that satisfy at least existential-mono-add allows us some liberty in our search for manipulation numbers. If we know that our voting rules satisfy existential-mono-add, we have a guarantee that extra manipulators beyond the minimum required cannot spoil a victory for a candidate. We escape having to verify the existence of a set of ballots for coalition sizes larger than the minimum, as well as having to check if a smaller coalition is capable of manipulation if we have confirmed that a larger one is not. The assumption of existential-mono-add allows us to use a non-linear search technique to compute manipulation numbers. Without assuming existential-mono-add, a search for manipulation numbers could only proceed linearly, because any search that skips intermediate values could not be sure of the optimality of any minimum it finds. For example, if a search skips to 10 manipulators and finds that such a coalition cannot make a certain candidate win, then with the assumption of existential-mono-add, it also can be sure that no coalition smaller than 10 could make that candidate win.

All the voting rules we examine in this work satisfy full mono-add-top,

except Copeland which does not even satisfy existential-mono-add.[1] For scoring protocols, it is obvious that since a unique winner holds the top score, adding additional ballots with that candidate at the top increases its score by at least as much as that of any other candidate. Woodall shows instant runoff voting (known as "alternative vote" in that work) satisfies mono-add-top [Woo94].

Plurality with two-candidate runoff satisfies it as long as candidates that are tied for second-most first-place votes in the first round are eliminated (which is how we have defined it in this work). Suppose candidate is already winning. Consider each of the possible score situations. If the candidate leads with multiple candidates tied for second place, all those candidates will still be eliminated if an extra point goes to the preferred candidate. If the candidate leads just one other candidate and wins after the transfers, adding a point to the candidate's score can't change who is eliminated in the first round, so the same transfers make the preferred candidate win. If the candidate shares the lead with one other candidate and again wins on transfers, then again the extra point cannot change the eliminated candidates, and the preferred candidate will remain the winner. If the candidate shares the lead with more than one other candidate, the extra point will cause the preferred candidate to win immediately. Finally, if the candidate trails one other candidate and wins on transfers, the extra point again cannot change the eliminations. If candidates tied for second are allowed to advance, then

[1] Consider the election

$$
\begin{aligned}
&a > b > c > d > v \\
&a > c > v > d > b \\
&d > a > b > c > v \\
&d > a > c > b > v \\
&b > c > d > a > v \\
&c > b > d > a > v
\end{aligned}
$$

a is currently the unique winner with three pairwise victories. b has one victory and two ties, c and d each have two victories and one tie, and v has no victories. Any additional ballot cast will break the ties between b and c, and b and d, resulting in an overall tie between a and one of b, c, or d.

the extra first round point can change the first-round eliminations and violate the mono-add-top.

Though Copeland does not satisfy existential-mono-add, we nevertheless found that a vast majority of elections generated according to the distributions described previously allowed additional manipulators to find appropriate additional ballots that would let a winning candidate remain a winner.

Theorem 2.1 (Non-reversion of manipulation for candidates)**.** If the manipulation number for a candidate c is k, and the voting rule satisfies existential-mono-add, then for all manipulating coalitions of size $k' > k$, that coalition can also make c a unique winner.

Proof. The first k manipulators can cast exactly the same ballots that they did to make c a unique winner and existential-mono-add guarantees the existence of ballots that the extra manipulators ($k' - k$ of them) can cast to keep c a unique winner. □

The next theorem establishes that the manipulation numbers for a manipulation instance do not decrease, and does not depend on mono-add-top. This property is used to save effort when we compute the full sequence of manipulation numbers for manipulation instances in the next chapter by allowing us to begin searches for subsequent manipulation numbers using the previous value as a lower bound.

Theorem 2.2 (Non-decreasing manipulation numbers for manipulation instances)**.** The manipulation numbers for any manipulation instance (C, V, R) are a non-decreasing sequence. Equivalently, if M_j for (C, V, R) is k, then M_{j-1} is at most k.

Proof. If the $M_j = k$, then the k manipulators can also make $j-1$ candidates unique winners. Thus, M_{j-1} is at most k. □

We can reexamine Conitzer and Sandholm's work on the frequency at which elections are manipulable in the context of manipulation numbers. First, Conitzer and Sandholm discard manipulation instances for which only

one candidate can be made to win. This is equivalent to discarding all instances for which the second manipulation number exceeds the size of the manipulating coalition. They remark that these instances are not interesting. In one sense this is true: When the goal is to highlight for a given k the ratio of instances for which $M_2 \leq k$ to those for which $k < M_3$, then indeed they are not interesting. But a broader examination of how often manipulation is hard for the coalition should include these cases. Indeed, if $k < M_2$, manipulation is not merely hard but impossible! Surely one must consider these cases.

Manipulation numbers are also interesting in another sense. In a way, they form a universal scoring system for elections. Some voting rules produce a score that conveniently summarizes an election, illustrating a margin of victory for a winner and how far each losing candidate was behind the winner. Scoring protocols and Copeland's rule are examples of rules that produce such a score. For some voting rules it is more difficult to summarize how well a losing candidate performed. Instant runoff voting, plurality-with-runoff, and the Schulze method [Sch03] are examples of rules that do not have a convenient score. Additionally, some definitions of criteria for voting rules are expressed more conveniently when the rule produces a score. A property known as *weak monotonicity* given in [CS06] is one such example. There, Conitzer and Sandholm phrase monotonicity for scoring protocols as "if a manipulator changes his vote so that a given candidate is ranked ahead of a larger set of candidates, then that candidate's score should not decrease." The general definition they give later is more complex.

Manipulation numbers can be used to provide a score to any candidate under any voting rule: The score is the minimum number of extra voters that would have to be added to unambiguously swing the election for a given candidate. Although they would have to vote ideally, it does provide a well-defined way for the additional voters to vote that has an intuitive rationale.

For example, in an election being decided by the instant runoff voting rule, one possible way of evaluating the losing candidates' performance or ranking the candidates would be to consider the elimination order when the

rule is executed. Consider an election with seven votes for $a > b > c$, four for $a > c > b$, nine for $b > a > c$, and 10 for $c > a > b$. In this election, b is eliminated first. The votes transfer to a, c is eliminated, and a wins. The ranking by elimination order is a (the non-eliminated winner), then c, and lastly b. However, both b and c can win with just four manipulators.[2] The manipulating coalitions would cast votes to raise their preferred candidate's count of first-place votes to 12, and then tie the other two candidates' first-place counts at 11. Both of the other candidates are eliminated together, and the coalition's preferred candidate wins. In this sense, b and c are actually tied.

This particular example highlights a disappointing property of instant runoff voting under the ties-eliminate model. In constructing the above example, we found that the most effective strategy for manipulating coalitions in instant runoff voting elections with the ties-eliminate model was to raise their candidate above the pack and force a tie among the other candidates. The transfer effect and candidates' support in non-first-place positions rarely had to be considered.

This possible application of manipulation as a scoring and summary mechanism is also another reason in support of the assumption of perfect information: Solving manipulation instances may not be an attack undertaken by malicious agents but rather a part of a post-election analysis to summarize candidate performances when the rule does not readily provide a convenient summary or score.

[2]Four was verified as the minimum coalition size for both candidates by our software, described in the next chapter.

Chapter 3

Computation of Manipulation Numbers

The primary contribution of this work to the field is an empirical analysis of the practical difficulty of manipulation. To aid in the empirical analysis, we used software techniques capable of computing the exact manipulation numbers of manipulation instances. For the purpose of this work we were not interested in the specific votes that the manipulators should cast to make a candidate win, and so as implemented we do not output the votes, but all the techniques we employ would be perfectly capable of producing the actual manipulator votes with only minor modification.

In the following sections we illustrate the techniques we used for computing manipulation numbers for a manipulation instance (C, V, R).

We give a procedure that will compute all the manipulation numbers for a manipulation instance (C, V, R) given an algorithm for computing the possible winner set for an instance (C, V, R, k).

Recall that the possible winner set is the set of all candidates who each separately could be made into a unique winner through the action of k manipulators. Each technique for computing the possible winner set is used as a black box in the procedure which computes manipulation numbers for (C, V, R). We chose to consider the methods of computing the possible winner set as a black box because there are numerous ways to either compute

or approximate the possible winner set of an instance (C, V, R, k), such as Bartholdi et al.'s GREEDY-MANIPULATION [BTT89a], Conitzer and Sandholm's FIND-TWO-WINNERS [CS06], the techniques presented here, as well as any others yet to be discovered.

We give two procedures that will compute the solution to the decision instance (C, V, R, c_p, k) (that is "Can c_p be made the unique winner of the election through the action of k manipulators?"). Each of these procedures is run on each candidate in C, to compute the possible winner set for an instance (C, V, R, k).

3.1 MANIPULATION-NUMBERS and FIND-FIRST

The following procedure computes the manipulation numbers for (C, V, R) using as a black box subroutine a procedure for computing the possible winner set from a manipulation instance (C, V, R, k).

```
MANIPULATION-NUMBERS(C, V, R)
1  N ← ⟨⟩
2  k ← 0
3  for i ← 1 to m
4      k ← FIND-FIRST(i, 0, 0)
5      Append k to N as the ith manipulation number.
6  return N
```

```
I-WINNERS(i, k)
1  return |POSSIBLE-WINNERS(C, V, R, k)| ≥ i
```

The procedure FIND-FIRST is a search procedure. It is a variation of binary search that searches for the first value k for which a predicate function is satisfied. As used here, it takes a parameter i which is passed through to the predicate I-WINNERS, an initial value to begin searching from, and a step size which is always zero for the top-level call.

The predicate function, in addition to receiving the parameter i (which remains the same throughout the tree of calls that implement a single

search), takes a single integer, and returns true or false. The FIND-FIRST procedure assumes that there is some integer k for which all inputs to the predicate less than k return false, and all inputs greater than or equal to k return true. If the predicate satisfies these assumptions, the FIND-FIRST procedure finds this value k, the first value that makes the predicate return true. For computing manipulation numbers, if the voting rule in use satisfies mono-add-top then we know that the size of the possible winner set never decreases as k is increased, so FIND-FIRST's requirement holds: there is some value (the ith manipulation number) below which the predicate will return false (there are fewer than i winners), and at or above this value the predicate will return true (there are i or more winners).

One feature of FIND-FIRST that distinguishes it from standard binary search is that FIND-FIRST does not require a known upper bound. It establishes an upper bound by evaluating the predicate at successively larger values, doubling the argument in each successive step. While we do in technically have an upper bound in this case, namely the point where the manipulators outnumber the non-manipulators, FIND-FIRST does not directly make use of that. That fact is used however by the POSSIBLE-WINNERS function—it returns all candidates as possible winners when the number of manipulators exceeds the number of non-manipulators.

FIND-FIRST also has the advantage in this application of tending to search smaller values first. This is advantageous when it is expected that smaller values of the predicate are quicker to compute, which we is the case here, and when the value that the predicate will return tends toward the lower end of any given interval, i.e., there is a higher probability that the value returned will be small. When computing manipulation numbers for elections generated under the uniform distribution, using FIND-FIRST rather than binary search between 0 and $n+1$ led to instances being solved in roughly five to fifteen percent less time. For elections drawn from the Condorcet and spatial distributions, the advantage was less distinct, but was not a hindrance.

Our software implementation made extensive use of caching wherever possible. Not only were the results of the predicate cached, but some eval-

uations of the predicate were eliminated entirely by using the assumption of mono-add-top in our voting rules. If in the search for minimum manipulation numbers a decision problem was evaluated, e.g., "Can a coalition of eight make a candidate a unique winner?" and later the search needed to decide if a coalition of size 10 could make that same candidate a unique winner, the caching mechanism would report in the affirmative without solving another potentially expensive decision problem, because a smaller coalition was already shown to be capable of this task. Similarly, if a given coalition was shown to be unable to make a candidate a unique winner, a smaller coalition was automatically assumed to be likewise unable to make to make that candidate a unique winner.

FIND-FIRST$(i, k, step)$

1 **if** I-WINNERS(i, k)
2 **return** 0
3 **if** I-WINNERS$(i, k + step)$
4 **return** $\lfloor \frac{step}{2} \rfloor$ + FIND-FIRST$(i, k + \lfloor \frac{step}{2} \rfloor, 0)$
5 **return** FIND-FIRST$(i, k, \max(1, 2 \cdot step))$

FIND-FIRST operates recursively. It assumes that once it has evaluated the predicate at a particular value, evaluating that value again is a fast operation, i.e., that the result is automatically memoized. The return value of any single call to FIND-FIRST is the offset from the parameter k of the lowest value to satisfy the predicate. The initial value of k in the call from MANIPULATION-NUMBERS is 0. If the predicate holds for 0, it returns 0. The procedure maintains the invariant that if the value k satisfies the predicate, it is the lowest value to do so. If k does not satisfy the predicate, it begins to search offsets from k of successive powers of two until it finds a value that satisfies the predicate. The value of the search offset is passed in the parameter *step*. When it finds a satisfying value, it begins the search again, with k advanced to the last value seen that does not satisfy the predicate, namely $\lfloor \frac{step}{2} \rfloor$, and step set to 0. When the recursive stack unwinds, the value of each frame's last unsatisfying search offset is added into a running sum. The end result of this sum is the lowest value to satisfy the predicate.

As an example of the operation of FIND-FIRST, if the smallest coalition size that can make a candidate win is 23, FIND-FIRST would evaluate the predicate for values of $k = 0, 1, 2, 4, 8, 16, 32, 17, 18, 20, 24, 21, 22, 24, 23$. Note how the search operates by doubling the offset from the last known lower bound, and resetting the offset each time a satisfying value is found. FIND-FIRST evaluates the predicate at 14 different values. A linear search would have evaluated 24. A standard binary search would need an upper bound to operate. If this election has, say, 128 non-manipulators, we can use an upper bound on the coalition size of 129. Binary search would have evaluated the predicate at $65, 33, 17, 26, 22, 24, 23$. At first, this appears more efficient. However, this sequence includes the value 65, which could be prohibitively expensive to compute. FIND-FIRST on this example never searches a value greater than 32. Also note that 24 appears in the list twice. The caching behavior in our implementation meant that the predicate was only evaluated at 24 once.

A bigger problem for binary search is what happens when the predicate cannot be evaluated within a given time limit. Because some instances of the predicate cannot be solved within a reasonable time limit, we attempt to find lower and upper bounds for manipulation numbers. We do this by running two separate searches. In the first search, if the predicate is not solved within the time limit, we assume that it returns true. In the second search, we assume it returns false. The result of this is that the two search procedures will give lower and upper bounds on the manipulation numbers, respectively. Previously, we assumed that the predicate always returned false up to a value k, then always returned true. When a time limit is added, this changes. Now, the predicate will return false up to a point, then there will be a region where instances will not be solvable within the time limit, followed by the region where it returns true. By assuming that the unknown region is false, the search procedure finds the first value for which the predicate is definitely known to be true. By assuming true in the unknown region, it finds the first value at which it could be true. These values form upper and lower bounds respectively on manipulation numbers.

Binary search when applied to larger instances where there was a larger unknown region would attempt to evaluate the predicate at more values in the unknown region. Since the unknown region is by definition the set of problem instances where the maximum allowable time limit was used, as a result, it performed worse overall in our experiments because of the amount of time it spent on harder instances that FIND-FIRST avoided. This more than offset the small amount of time gained on smaller instances.

3.2 Brute force

The simplest technique we used to solve decision manipulation instances (C, V, R, c_p, k) was a naïve brute force search over all possible manipulator vote multisets. We enumerated every possible multiset of manipulator votes, then combined each such multiset with the votes of the non-manipulating voters and applied the voting rule to the combined set of votes. The manipulator votes were enumerated by enumerating all lists of $m!$ non-negative elements that sum to k. We enumerated these lists with a variation of integer partitions [KS98]. Unlike standard integer partitions where the parts are strictly non-increasing and zeros are disallowed, the order of the parts of the partition was retained, zero was allowed as a valid element of the partitions, and we ensured each partition had $m!$ entries.

The following pseudo-code procedure generates all the distinct lists of $m!$ non-negative elements that sum to k. These lists correspond precisely to the set of *all* the possible ways k manipulators could cast their ballots for m candidates.

```
MANIPULATOR-VOTES(k, length)
 1  if k = 0
 2      return {⟨0, 0, ..., 0⟩} ▷ list has length elements.
 3  if length = 1
 4      return {⟨k⟩}
 5  x ← ∅
 6  for c ← 0 to k
 7      x' ← MANIPULATOR-VOTES(k − c, length −1)
 8      for i ← 0 to |x'| − 1
 9            Place c onto the front of x'_i
10      x ← x ∪ x'
11  return x
```

The parameter k is the value each returned list should sum to, and is also the number of manipulator votes that remain undetermined at each recursive step. The parameter *length* is the how long the resulting lists should be. For the first call, its value is $m!$, giving a list that has one entry for each lexicographic permutation of the candidates. Each resulting list is interpreted as a set of manipulator votes.

The above function has two base cases. When the number of undetermined manipulator votes is zero, all remaining positions in the list are filled with zeros. When the *length* parameter gets down to one, meaning there is only one entry in the list undetermined, the returned list has a single entry containing all the remaining manipulator votes. After checking for base cases, the recursive case is executed. For each possible number of manipulator votes to be assigned to the next position, c, the procedure calls itself recursively and gets the set of all lists having c fewer manipulator votes (since it is about to assign c of them to the current position), and one fewer position to be filled. c is then attached to the front of each list returned by the recursive call. All values of c from 0 to all of the remaining manipulators are valid values for the current entry of the list. If we were computing the answer to an instance (C, V, R, c_p, k) where $k = 4$ and $m = |C| = 3$, the above procedure would be called with $k = 4$ and $length = m! = 6$, and

it would give a set of lists. Each list would have six non-negative integer elements, and each list would sum to four. Each list would represent one way the manipulators could vote.

The ith integer in each final list in the returned set corresponds to the number of times that the ith lexicographical permutation of the candidates should be included as a vote. We used a simple combinatorial UNRANK procedure from [KS98] to get a specified lexicographic permutation of a set of objects. The parameter *objects* is initially a list of the candidates in some distinguished initial order. The parameter *rank* specifies which lexicographic permutation of the candidates we want.

```
UNRANK(objects, rank)
1  if | objects | = 1
2      return objects
3  position ← ⌊rank / b⌋
4  b ← (| objects | − 1)!
5  rest ← UNRANK(TAIL(objects), rank mod b)
   ▷ TAIL returns its argument with the first element removed.
6  Return rest with objects[0] inserted at index position.
```

Using the example from above with $k = 4$ and $m = 3$, one of the lists in the returned set would be $\langle 1, 0, 1, 1, 0, 1\rangle$, and it would correspond to the set of votes $\{a > b > c, b > a > c, b > c > a, c > b > a\}$—one vote each in each of the first, third, fourth, and sixth lexicographic permutations of the candidates.

To compute the possible winner set, we applied the voting rule to the concatenation of the non-manipulators' votes and each possible multiset of manipulator votes. If the rule produced a unique winner, it was added to the possible winner set. The procedure MAKE-VOTES takes as parameters the candidate set and one of the lists produced by MANIPULATOR-VOTES, and prepares a multiset of votes by adding $list_j$ instances of the jth lexicographic permutation of the candidates.

```
POSSIBLE-WINNERS-BY-BRUTE-FORCE(C, V, R, k)
1  winners ← ∅
2  votes ← MANIPULATOR-VOTES(k, |C|!)
3  for i ← 0 to | votes | − 1
4      M ← MAKE-VOTES(C, votes_i)
5      W ← R(C, V ∪ M)
6      if |W| = 1
7          winners ← winners ∪W
8  return winners
```

Any election that produced multiple winners was treated as having no winner, as this work only concerns manipulators making a candidate the unique winner. We stored the running set of candidates for which the search had found a set of manipulator votes that could make that candidate a unique winner. The set of unique winners seen after examining all the manipulator vote profiles was taken as the set of possible winners. As a time-saving measure not illustrated in the code above, the process was stopped early whenever the set of possible unique winners contained all the candidates. This naturally implies the process was never stopped early when some candidates could not be made unique winners by the efforts of the manipulators.

This technique was deliberately naïve. It represented the simplest possible method that would yield a correct answer. It was useful to get a baseline for comparison of the efficiency of other techniques, and also to test the correctness of our implementation of other techniques.

3.3 Reductions and state-of-the-art solvers

Our main alternative method for computing the possible winner set for manipulation instances (C, V, R, k) works by computing the answers to each element of the set of m decision problems $\{(C, V, R, c, k) \mid c \in C\}$. We solved these decision problems by reducing them to other problems which are better understood in the literature, namely 0–1 integer programming and

CNF-SAT. CNF-SAT was the first problem shown to be NP-complete, by Cook [Coo71]. Shortly after, 0–1 integer programming was shown to be NP-complete by Karp as one of his twenty-one NP-complete problems [Kar72].

0–1 integer programming is a special case of integer linear programming where the values of the variables are restricted to 0 or 1. Integer linear programming is also NP-complete, and is often used as a reduction target for other NP-complete problems [Pap81, WN99]. If the variables are not restricted to the integers, the problem is known as linear programming. Linear programming is not NP-complete and efficient techniques for solving linear programs are well-studied [Mur83].

An instance of integer linear programming consists of a set of inequalities in several variables, often denoted as a series x_i. The inequalities are all linear in the variables x_i—they consist of a sum of terms on the left side, and a constant on the right side, and each term includes only one variable to the first power, multiplied by a constant coefficient. The inequalities have the general form

$$\sum_i a_i \cdot x_i \leq b$$

The instance asks if there is an assignment of integer values to the variables x_i that satisfies all of the inequalities. A simple example of an integer linear program would be the set of inequalities

$$\begin{aligned} 2x_1 &\leq 9 \\ -x_1 &\leq -3 \end{aligned}$$

There exists a solution to this set of inequalities at, e.g., $x_1 = 4$, so the instance is satisfiable.

Some definitions of integer linear programming allow requiring the minimization or maximization of some objective, e.g., find an assignment of the variables that satisfies all of the inequalities and for which the value of x_1 is minimized over all satisfying assignments. In the above example, the minimized solution is $x_1 = 3$. Since our problems are decision problems,

we are not interested in minimization or maximization, only whether or not the instances are satisfiable. In the reductions we have come up with for the decision instances, we do not need the variables of the inequalities to assume any value other than 0 or 1.

CNF-SAT is a decision problem that asks "Given a Boolean formula in conjunctive normal form, is there an assignment of truth values to the variables of the formula that makes the entire formula true?" A Boolean formula in conjunctive normal form is a single conjunction of disjuncts. Each disjunct is a set of disjoined literals, and each literal is either a variable or the negation of a variable.

CNF-SAT is a special case of 0–1 integer programming. One can see this by letting the variables of the CNF-SAT formula correspond with variables in 0–1 integer programming. Each conjunct c becomes the inequality

$$\sum_{\ell \in c} \text{coeff}(\ell) \cdot \text{var}(\ell) \geq 1 - \#\text{negated literals in } c$$

where $\text{coeff}(\ell)$ is -1 if ℓ is a negated literal and 1 otherwise, and $\text{var}(\ell)$ is a 0–1 integer programming variable corresponding to the variable in ℓ. The conjunction of the disjuncts in CNF-SAT is mirrored by the requirement in 0–1 integer programming that all the inequalities be simultaneously satisfied.

As an example of this equivalence, consider the CNF-SAT formula

$$\neg x_2 \vee x_4$$
$$\neg x_3 \vee \neg x_6$$

According to the above transformation, this yields the inequalities

$$-x_2 + x_4 \geq 0$$
$$-x_3 - x_6 \geq -1$$

Under this scheme there is a direct correspondence between CNF-SAT variables and 0–1 integer programming variables. For convenience in writing

the reductions, we sometimes use the notation of CNF-SAT alongside the notation for 0–1 integer programming. Thus, we may choose to write

$$x_1 + x_2 + x_3 - x_4 - x_5 - x_6 \leq 0$$
$$\neg x_2 \vee x_4$$
$$\neg x_3 \vee \neg x_6$$

This should be clearer than if the second two forms were written as the inequalities $-x_2 + x_4 \geq 0$ and $-x_3 - x_6 \geq -1$. In either case, it is seen that there is no real change in the problem being hidden by notation.

We selected 0–1 integer programming as a reduction target because it seemed particularly well-suited for constructing reductions from the manipulation problems for the voting rules we examine. Manipulation for voting rules we examined can all be phrased conveniently using constraints allowed in 0–1 integer programming and CNF-SAT. This combination of 0–1 integer programming and SATISFIABILITY is sometimes called pseudo-Boolean satisfiability, or pseudo-Boolean optimization. For example, plurality voting can be expressed as a collection of inequalities, one for every candidate other than the one the manipulators prefer, each asserting that the number of first-place votes the preferred candidate received is strictly greater than the number for the other candidate. The number of first-place votes a candidate received is computed from an encoding of the ballots. If $F_{v,c}$ is a variable representing that voter v put candidate c in first position (F for first position), comparing the number of first place votes for two candidates a and b and ensuring that a has more than b is a single inequality

$$\sum_{v \in V} F_{v,a} - \sum_{v \in V} F_{v,b} \geq 1$$

Our encoding of the ballots for manipulation of plurality voting (and of all scoring protocols) has a variable for each candidate-ballot pair that is 1 iff that candidate is in the first position on that ballot (as well as variables for the other positions). The votes for the non-manipulators are specified by adding single literal clauses to set the truth value of the variables rep-

resenting their votes. The manipulators' ballots are left free to be solved by the solver, but constraints are placed upon them to make sure the solver does not assign illegal ballots to the manipulators in its search for an assignment to the variables. It is here that expressing elements of the reductions as CNF-SAT is handy. Many of these constraints take the form of simple implications, and it would be awkward to write these constraints only in inequalities.

Here is an example of one such set of constraints placed on the manipulator ballots. If a voter has a candidate in first-place, no other candidate can be in first place.

$$\bigwedge_{\substack{v \in V \\ c,d \in C, c \neq d}} F_{v,c} \rightarrow \neg F_{v,d}$$

Although manipulation for plurality voting is trivial, it demonstrates the convenience of using 0–1 integer programming with CNF-SAT notation as a reduction target. We find that many voting rules incorporate operations like taking sums of variables, and constraining the possible values of the sums with inequalities. Thus we expect that 0–1 integer programming would be a useful target for constructing reductions for other voting rules that we do not examine.

After manipulation instances were reduced to 0–1 integer programming problems, we solved them using state-of-the-art solvers for integer linear programming and CNF-SAT. To get CNF-SAT instances from 0–1 integer programming instances we used a reduction described by Warners [War98]. This reduction is described in Section 3.3.2.

Our initial motivation for exploring reductions was to allow the solving of manipulation instances by state-of-the-art solvers. Since the main concern of this work is to explore the practical difficulty of manipulation, state-of-the-art solvers were a natural choice, as they represent the culmination of great amounts of research into solving these well-known NP-complete problems efficiently [DP60, DLL62, BHZ06]. State-of-the-art solvers are capable of solving instances far beyond the range of tractability of a brute force search [LZD04]. Moreover, the state-of-the-art in these well-known

problems continues to develop, so reductions that target these problems will inherit progress made in solver research. There is an annual conference on practical applications of SATISFIABILITY [SS10], that has in recent years been accompanied by a competition between SATISFIABILITY solvers to see which can solve the most instances from a battery of large instances, and minimize the time they take to do so [SP08, BRS09].

By reducing to 0–1 integer programming and CNF-SAT we were able to solve larger instances than would have been possible using only a brute force method. We observed that when computing manipulation numbers, solvers were able to quickly determine unsatisfiability for manipulation instances with few manipulators, and to quickly show satisfiability for manipulation instances with many more manipulators than the minimum needed to make a candidate a unique winner. We observed there was a core of instances where the number of manipulators was close to the minimum needed where the instances grew very difficult to solve, growing harder for the solvers to solve closer to the exact minimum. While some of the instances were not solvable within a given time limit, if one is willing to accept an approximation, our technique of computing manipulation numbers gives an upper and a lower bound for manipulation numbers which could not be exactly computed within the time limit, and the precision of these estimates can be increased by increasing the time limit.

Our system of reducing manipulation problems to 0–1 integer programming and CNF-SAT is capable of producing an actual set of manipulator votes that would make the preferred candidate win when such a set of votes exists, i.e., it can produce the witnessing set of votes (not merely assert that they exist) as long as the solver used produces a satisfying model for instances it determines to be satisfiable.

It is also relatively simple to add support for additional voting rules. The details of the voting rule need only be constructed using a number of easily-combined constructs we have written for describing voting rules in CNF-SAT. We have these common, easily-combined voting rule constructs defined in our software, which we describe in Section 3.3.4.

3.3.1 State-of-the-art solvers used

We used the GNU Linear Programming Kit [Mak] to solve 0–1 integer programming instances, and Clasp [GKS10] to solve CNF-SAT instances. The GNU Linear Programming Kit was used because it was a freely available 0–1 integer programming solver. Clasp was selected for its gold-medal performance in the crafted category in the 2009 SAT competition [BRS09].

One may wonder why we look at both 0–1 integer programming and CNF-SAT as final targets for our reductions. It would initially seem that the reduction to 0–1 integer programming is more direct, and yields smaller problems with far fewer total constraints. Intuition would also seem to suggest that being able to use the hardware adders on a CPU would give an advantage to a 0–1 integer programming solver over a CNF-SAT solver that is implementing adders via constraints in the formula. However Li et al. have shown that SATISFIABILITY solvers can be very efficient at solving integer programming problems [LZD04].

To compare the GNU Linear Programming Kit and the CNF-SAT solvers, we reduced manipulation problems to CNF-SAT by applying Warners' reduction to the 0–1 integer programming elements, and also reduced the same instances to 0–1 integer programming by converting the CNF-SAT elements to 0–1 integer programming. In these early experiments, Clasp applied to the CNF-SAT forms far outperformed the GNU Linear Programming Kit on the 0–1 integer programming forms, by at least an order of magnitude. A possible explanation for this is that some of the constraints resulting from reducing manipulation yielded problems that had a large number of implications. All the reductions use many implications to specify what is a valid vote for a manipulator to cast—if some condition is true about a ballot (a candidate is given a certain rank) then another condition must be false (no other candidates may be given that rank). When these are converted to inequalities, the implications may not be as readily detectable by the solver. Similarly, other parts of the reductions are very readily expressed as logical Boolean formulas, and comparatively fewer parts were expressed as inequalities. While reducing inequalities to CNF-

SAT may seem more awkward, more of the constraints were expressed as Boolean formulas. These factors probably lead to Clasp performing far better on these problems than the GNU Linear Programming Kit. This explanation agrees with the observations of Manquinho et al. in [MMSOS98], that 0-1 integer programs where the models are very hard to satisfy presents trouble for branch-and-bound based solvers, as they can enter branches of the search tree where the model is not satisfiable at all and they take too long to determine this and back out.

3.3.2 A reduction from 0–1 integer programming to CNF-SAT

As stated previously, our manipulation instances may contain both CNF-SAT clauses and 0–1 integer programming inequalities. To arrive at a single formula that is only CNF-SAT, the inequalities must be turned into CNF-SAT formulas. We now present the technique we used to turn the 0–1 integer programming inequalities in our manipulation instances into CNF-SAT formulas.

To turn 0–1 integer programming into CNF-SAT, we need to convert the inequalities into Boolean formulas that have the same meaning. We did this by using the transformation described by Warners [War98], which produces a formula that is linear in the number of summands in the inequality.

Warners' transformation begins by computing the sum of the left side of the inequality. Computing the sum of the left side begins with computing the value of each summand. Observe that each summand is actually a coefficient multiplied with a variable, and that variable is restricted to 0 or 1. The left side has the form

$$\sum_i a_i \cdot x_i$$

Each summand $a_i \cdot x_i$ is represented by a vector of CNF-SAT variables, representing the binary encoding of the integral value of the result of the multiplication. Let z_i be the vector of variables representing the value of the ith term.

The truth values of the z_i vector are uniquely determined by the value of a_i and the truth value of variable x_i. The value of a_i is known at reduction time, but the value of x_i is not. There are only two possible states for x_i, and hence there are only two possible states for the entire vector z_i. If x_i is false, all variables for in the vector are false, making the summand zero. If x_i is true, those variables corresponding to bits that are true in the binary representation of the integer coefficient a_i are true, and the others are false, giving the summand the value of a_i. For example, if the value of a_i is 9, with the binary representation 1001_2, the resulting formula would force $z_{i,1}$ and $Z_{i,2}$ to be false always, and $z_{i,0}$ and $z_{i,3}$ to be true if and only if x_i is true. Thus, the formula would contain

$$\neg z_{i,0} \vee x_i$$
$$z_{i,0} \vee \neg x_i$$
$$\neg z_{i,1}$$
$$\neg z_{i,2}$$
$$\neg z_{i,3} \vee x_i$$
$$z_{i,3} \vee \neg x_i$$

If a_i is negative, a simple algebraic manipulation is employed to make an equivalent inequality with only positive coefficients. For any negative a_i, $|a_i|$ is used in its place, $|a_i|$ is added to the value of the right side of the inequality, and x_i is replaced with $\neg x_i$. For example, the inequality $-3x_0 + -7x_1 + x_2 \leq 0$ is equivalent to $3(\neg x_0) + 7(\neg x_1) + x_2 \leq 10$.

After the "multiplication" step, there are now several vectors of literals, each one encoding an integer that is part of the left side sum. Warners' transformation implements a binary adder for each pair of summands, and the result is another collection of variable vectors, now representing the sum of pairs of terms. This process of adding pairs is repeated until there is a single vector of variables encoding the sum of the entire left side. If there are an odd number of terms in any of these rounds, the case is handled by leaving the unpaired term unchanged, and carrying it forward into the next

round.

The final comparison between the sum of the left side of the inequality and the value on the right side of the inequality is performed with a binary less-than-or-equal-to operation. Since the inequalities are defined to have a constant on the right side, and the adder structure from the left side ends with the sum of the left-hand side as a binary-encoded integer in a vector of variables, the comparison constraint merely has to assert that the latter vector encodes a value less than or equal to the former vector.

For convenience, let us call the left side vector x and the right side vector y. Observe that if $x > y$, then for some bit position k, the kth bit of x is true, while the kth bit of y is false and there is no more significant bit position where the opposite relation holds, i.e., there is no $j > k$ (bit j is more significant than bit k) such that the jth bit of x is false, while the jth bit of y is true. Thus, to assert that $x \leq y$, we assert for any bit position that is true in x and false in y that there *must be* a more significant bit position that is false in x and true in y. To show it more succinctly in conjunctive normal form, we borrow a piece of notation used by Warners when describing the constraint. Let B_y be the set of bit indices in y that are true. All indices are assumed to be between 0 and the length of the bit vector representing each number, so that they are valid indices.

$$\bigwedge_{k \notin B_y} \left(\neg x_k \vee \bigvee_{j \in B_y, j > k} \neg x_j \right)$$

3.3.3 Embedding nested constraints

We also employ in our reductions the well-known technique of converting a general Satisfiability formula into a CNF-SAT formula that is equisatisfiable. For example, the non-CNF formula

$$(x_1 \wedge y_1) \vee (x_2 \wedge y_2)$$

may be converted into a CNF-SAT formula by introducing a new surrogate variable for each sub-expression which is in conjunctive normal form,

and forcing this variable to be logically equivalent to its associated sub-expression. This is known as a Tseitin transformation or reifying the expression [Tse68, BHZ06].

In this example, we would introduce S_1 for $x_1 \wedge y_1$ and S_2 for $x_2 \wedge y_2$, then add clauses to the formula to make them logically equivalent, yielding the formula

$$S_1 \vee S_2$$
$$S_1 \leftrightarrow (x_1 \wedge y_1)$$
$$S_2 \leftrightarrow (x_2 \wedge y_2)$$

which can be reduced to the CNF-SAT formula

$$S_1 \vee S_2$$
$$\neg S_1 \vee x_1$$
$$\neg S_1 \vee y_1$$
$$\neg x_1 \vee \neg y_1 \vee S_1$$
$$\neg S_2 \vee x_2$$
$$\neg S_2 \vee y_2$$
$$\neg x_2 \vee \neg y_2 \vee S_2$$

To generalize this process, let us now assume we have a CNF-SAT sub-expression ϕ, and we wish to introduce a surrogate for ϕ, S_ϕ.[1]

S_ϕ is a new variable. We need merely to enforce $\phi \leftrightarrow S_\phi$, and then use

[1] If ϕ is not in CNF, perform this procedure recursively on the sub-expressions of ϕ. Note however that if a sub-expression of ϕ, call it ρ, is recursively embedded, it is vital that the clauses that enforce the relationship between ρ and S_ρ not be included in as part of the original expression ϕ. Otherwise, S_ϕ and ϕ would not be equisatisfiable: A trivial satisfying assignment could be found by violating one of the clauses enforcing $\rho \leftrightarrow S_\rho$. All clauses generated by any recursive applications of this process must be placed in the top-level formula.

S_ϕ in place of ϕ in the original formula. The simpler half is $S_\phi \rightarrow \phi$. Since ϕ is a Boolean formula in conjunctive normal form, we can distribute the implication over the conjuncts, and thus we need only one constraint for each conjunct c in ϕ.

$$\bigwedge_{c \in \phi} \neg S_\phi \vee c$$

To enforce the other implication, $\phi \rightarrow S_\phi$, note still that ϕ is a collection of conjuncts. The structure we have is

$$\left(\bigwedge_{c \in \phi} c \right) \rightarrow S_\phi$$

To require that all conjuncts c be true to imply S_ϕ, we must introduce a new surrogate for each conjunct c, S_c and replace the conjuncts c in the above CNF by their corresponding surrogates S_c. As long as we also enforce $\bigwedge_{c \in \phi}(S_c \leftrightarrow c)$, this completes the embedding of ϕ: We may use the variable S_ϕ in place of ϕ.

To enforce the condition $S_c \leftrightarrow c$, we add clauses for $S_c \rightarrow c$ and $c \rightarrow S_c$ to the top-level formula for each c. Since c is a collection of disjuncts, $S_c \rightarrow c$ is given by

$$\neg S_c \vee c$$

Likewise, $c \rightarrow S_c$ is given by $\bigwedge_{d \in c} d \rightarrow S_c$, or

$$\bigwedge_{d \in c} (\neg d \vee S_c)$$

for each disjunct d in c.

If we wish to embed inequalities using this technique, they are first reduced to CNF-SAT using Warners' reduction, and the resulting formula embedded. However, it is very important to only perform the final Tseitin transformation on the clauses that implement the comparison of the final bit vectors. This is because we want the new variable introduced by the transformation to be equivalent to the actual comparison of the numbers being

compared. The clauses that in the "multiplication" and "addition" steps implement arithmetic that computes the value of the left-hand side of the inequality. If these clauses were included in the Tseitin transformation of the inequality, the resulting formula would be satisfiable by simply violating the rules of arithmetic when computing the value of the left-hand side.

3.3.4 Common voting rule reduction elements

There are common elements that we observed when constructing reductions for the manipulation problem of multiple voting rules. Many voting rules employ similar concepts such as pairwise comparison of candidates, eliminating candidates who did poorly from further consideration, and awarding a score to candidates based on their raw performance on the ballots and then comparing those scores.

We engineered our reductions so that we could combine components that implemented common elements seen in voting rules, allowing us to check smaller components for correctness more easily, as well as simplify the task of building reductions for new voting rules in the future.

Ballots

The first common task to consider is a representation of the ballots of the voters. We have two representations that we use in our reductions. The first representation stores the exact position in which a voter placed each candidate on their ballot. We call this the positional ballot scheme. In this scheme, we define a variable in our formula for each voter v, candidate c, and position p, giving $V_{v,c,p}$. Positions are integers from 0 to $m-1$. Variable $V_{v,c,p}$ is true iff voter v ranked candidate c in position p. This is done for every voter, manipulator and non-manipulator alike. The votes of the non-manipulators are very simple to encode. We use single variable clauses to set the state of variables for which we know the truth value. A reduction using the positional ballot scheme produces clause

$$V_{v,c,p}$$

for all v, c, p, if non-manipulator v put candidate c in position p, and clause

$$\neg V_{v,c,p}$$

if non-manipulator v ranked candidate c in a position other than p. This information is available at reduction-time, specified by the votes in the set V, the vote profile of the election.

The votes of the manipulators are going to be determined by the solver, so we must force the solver to assign valid votes to the manipulators. There are three major sets of constraints to place constraints on how each manipulator votes.

- Ensure that each manipulator does not rank two candidates in the same position.
- Ensure that each manipulator ranks every candidate in some position.
- Ensure that no manipulator ranks a candidate in more than one position.

We must force manipulators never to put two candidates in the same position. Such a constraint is straightforward to encode in CNF-SAT, so this is done through a simple enumeration of the individual constraints. For each manipulating voter v, distinct position p, and distinct pair of candidates c and d, we introduce the clause

$$\neg V_{v,c,p} \vee \neg V_{v,d,p}$$

We must also force every candidate to be in some position. For every manipulating voter and candidate we have the clause

$$\bigvee_{p \in \{0,\ldots,m-1\}} V_{v,c,p}$$

There is no explicit provision to prevent a manipulator from placing a candidate in more than one position. The previous two constraints form a

bijection between each manipulator's ballot positions and the candidates, which implies that no candidate is in more than one position on any ballot. We felt that the redundant clauses would enlarge the formulas to be solved unnecessarily. Recent evidence suggests however that adding redundant clauses to formulas when they are not strictly needed can lead to solvers solving the formulas more quickly [Heu09]. This could be investigated further.

The second representation we use in our reductions is the pairwise ballot scheme. In this scheme, instead of encoding positions directly, we encode, for each pair of candidates, which one outranks the other. For each voter and pair of candidates, we have $P_{v,c,d}$, which is true iff voter v ranks candidate c above candidate d. The number of variables required is asymptotically identical to the positional ballot scheme.

As before, encoding the non-manipulators' votes is accomplished with collection of single literal clauses that sets the state of the variables for the non-manipulators' ballots. We add the single literal clause

$$P_{v,c,d}$$

if candidate c is above candidate d on non-manipulator v's ballot and

$$\neg P_{v,c,d}$$

otherwise. Also, for all candidates c we add

$$\neg P_{v,c,c}$$

To ensure the manipulators' ballots are valid, we have three sets of constraints to apply.

- Ensure exactly one of $P_{v,c,d}$ and $P_{v,d,c}$ is true, for all distinct candidates c and d.
- Ensure $P_{v,c,c}$ is false for all candidates.
- Ensure $P_{v,c,d} \wedge P_{v,d,e} \rightarrow P_{v,c,e}$ for all distinct candidates c, d and e.

The first set enforces asymmetry. The second enforces irreflexivity. The third enforces transitivity. We can rephrase the first two statements above into equivalent statements that are better for generating clauses in conjunctive normal form. These are

- Ensure at least one of $P_{v,c,d}$ and $P_{v,d,c}$ is false, for all candidates c and d.
- Ensure at least one of $P_{v,c,d}$ and $P_{v,d,c}$ is true, for all distinct candidates c and d.

Note that one of these statements includes the word "distinct" and the other does not. These statements are encoded in the formula as

$$\left(\bigwedge_{\substack{v \in M \\ c,d \in C}} \neg P_{v,c,d} \vee \neg P_{v,d,c} \right) \wedge \left(\bigwedge_{\substack{v \in M \\ c,d \in C, c \neq d}} P_{v,c,d} \vee P_{v,d,c} \right)$$

The third statement is encoded in the formula by

$$\bigwedge_{\substack{v \in M \\ c,d,e \in C \\ c \neq d, c \neq e, d \neq e}} \neg P_{v,c,d} \vee \neg P_{v,d,e} \vee P_{v,c,e}$$

In each of our reductions of manipulation for various voting rules we will use one of these two ballot schemes.

Eliminations

Some voting rules, including instant runoff voting and plurality with two-candidate runoff in this work, have a concept of elimination. Candidates who do not perform well in early rounds are eliminated. Candidates who are eliminated are completely removed from further consideration. They are, for all purposes, no longer on any of the ballots, and once eliminated, a candidate can never return.

Our reductions for these two voting rules use a common encoding for this concept of elimination. Elections in which eliminations occur always

proceed in a series of rounds.

Depending on the voting rule, the number of rounds used will vary, so we will let the set $\mathcal{R}$ be the set of rounds. For instant runoff voting, $\mathcal{R}$ is $\{0, \ldots, m-1\}$ and for plurality with two-candidate runoff, $\mathcal{R}$ is $\{0, 1, 2\}$. Round 0 is the initial state of the election, with no candidates eliminated. The rule may then decide to eliminate one or more candidates, and those candidates' elimination statuses are set for round 1.

We use a variable $E_{r,c}$ to represent elimination. $E_{r,c}$ is true if and only if candidate c is eliminated for round r. Since all candidates are in the election to begin, we have

$$\bigwedge_{c \in C} \neg E_{0,c}$$

Once a candidate is eliminated, that candidate cannot return. This next set of clauses cascades the elimination status for a given candidate through to subsequent rounds.

$$\bigwedge_{\substack{c \in C \\ r \in \mathcal{R} - \{0, \max \mathcal{R}\}}} \neg E_{r,c} \vee E_{r+1,c}$$

We only enumerate this clause over the "inner" rounds of $\mathcal{R}$, excluding round 0 and the last round, because no candidates have been eliminated in round 0, and the last round has no subsequent round, so elimination statuses of that round cannot and do not need to be carried anywhere.

The reductions for individual voting rules will specify further how candidates become eliminated, and how later rounds are evaluated when some of the candidates have been eliminated.

Scores

Several of the reductions involve computing some intermediate result and tallying up a score for each candidate. If we only used 0–1 integer programming inequalities as black boxes, this would cause much duplication in the resulting formulas. In a scoring protocol, comparing candidates' total scores requires a separate 0–1 integer programming inequality for each comparison, each one having all the logic for multiplying coefficients by their indicator

variables and for summing the summands. For example,

$$\left(\sum_{\substack{v\in V\cup M\\ p\in\{0,\ldots,m-1\}}} \alpha_p \cdot V_{v,d,p}\right) - \left(\sum_{\substack{v\in V\cup M\\ p\in\{0,\ldots,m-1\}}} \alpha_p \cdot V_{v,c,p}\right) \leq -1$$

this is the 0–1 integer programming inequality that would be needed to compare the scores of two candidates c and d, and ensure that c outscored d. Having such an inequality for each non-preferred candidate would mean that the clauses to compute the preferred candidate's score would appear once for every such inequality, not only giving an unnecessarily larger formula, but also obscuring some information that might be helpful to a SATISFIABILITY solver by having duplicated variables that would always have the same value, but having no explicit relationship in the formula.

To avoid this waste, we enhanced Warners' reduction by taking each of the fundamental operations used and allowing them to be applied separately. We use this technique specially in our reductions to CNF-SAT. Candidate scores are computed once, and are then referenced as needed in other parts of the reduction. Applied to the example above, this means we place clauses in the formula to compute each candidate's total points awarded by the scoring protocol, giving a bit vector representing each candidate's score. We can then apply the comparison step separately, essentially

$$\text{Score}_c = \sum_{\substack{v\in V\cup M\\ p\in\{0,\ldots,m-1\}}} \alpha_p \cdot V_{v,c,p}$$

for each candidate c, followed by

$$\text{Score}_d - \text{Score}_c \leq -1$$

to ensure c outscores d.

3.3.5 Reductions from manipulation to 0–1 integer programming

We now describe the individual reductions of manipulation instances to 0–1 integer programming, making use of the notation of CNF-SAT in some places, for convenience, and applying the optimization described in Section 3.3.4 for scores when reducing down to CNF-SAT. There are separate descriptions for each voting rule.

Scoring protocols

Because of the similarity in all voting rules that are scoring protocols, we were able to define a single reduction for all rules of this family. For scoring protocols we use the positional ballot scheme described in Section 3.3.4.

The positional ballot scheme makes it very simple to define the rest of the reduction. All we have to do is use the information contained in the variables that encode the ballots to calculate each candidate's total score, then assert that the preferred candidate's score is at least one point greater than that of each other candidate.

Let α be the score vector for the scoring protocol manipulation problem we are reducing. By applying Warners' summation method, we easily build up each candidate c's score in a bit vector

$$\text{Score}_c = \sum_{\substack{v \in V \cup M \\ p \in \{0,\ldots,m-1\}}} \alpha_p \cdot V_{v,c,p}$$

then for each non-preferred candidate d and the preferred candidate c compare the difference between the scores to the constant -1

$$\text{Score}_d - \text{Score}_c \leq -1$$

The resulting difference must be less than or equal to -1. This ensures the preferred candidate has defeated every other candidate by at least one point and is the unique winner.

An alternate scoring protocol reduction

An interesting alternative representation for this problem, due to Chamberlin [Cha85], is to encode the problem in $m!$ integers representing the number of voters that cast their ballot for each permutation of the candidates. In other words, encoding just the numerical voting profile. For example, with three candidates a, b, and c, we have six integers, representing the number of voters who vote for each permutation, $a > b > c$, $a > c > b$, $b > a > c$, $b > c > a$, $c > a > b$, and $c > b > a$, respectively. To encode the non-manipulators' votes, we would specify minimums on each of these variables:

$$x_i \geq n_i$$

where n_i is the number of non-manipulators who ordered the candidates in the ith permutation.

The number of manipulators is specified by encoding total number of votes:

$$\sum_i x_i = n + \#\text{manipulators}$$

Lastly the condition that the preferred candidate be the uniquely victorious could be expressed through $m - 1$ inequalities. The coefficients applied to the variables are determined by inspecting the order of the candidates in the ith permutation, and calculating the point advantage that a vote so cast gives to the preferred candidate. Below is an example using the Borda count. Supposing that the preferred candidate is b, b must defeat a,

$$x_1 + 2x_2 - x_3 - 2x_4 + x_5 - x_6 \leq 0$$

and b must defeat c,

$$-x_1 + x_2 - 2x_3 - x_4 + 2x_5 + x_6 \leq 0$$

While this representation is simpler and yields a much simpler reduction for smaller numbers of candidates, we nonetheless feel it has shortcomings. It has a $O(m!)$ growth in the number of candidates in the model, whereas

the representation we have used is $O(m^2)$ in the number of candidates. We also feel that it is more difficult to adapt this representation to other voting rules. Consider the instant runoff voting rule. Since this voting rule eliminates candidates who score poorly, to calculate scores in successive rounds would require conditionally selecting new coefficients for the variables x_i, depending on which candidates are eliminated. In his article Chamberlin did show results for the instant runoff voting rule (known as Hare voting in that article). If the coefficients were computed at reduction time, which we believe must be the case as we see no way calculate the coefficients within the reduction while retaining its simplicity, there would have to be alternate inequalities in the model for every possible set of eliminated candidates. This gives a $O(2^m \cdot m!)$ growth of the model in the best case (recall too that each inequality is of size $m!$). This is why we opted not to use this representation.

Instant runoff voting

In instant runoff voting, the next candidate to be eliminated in each round is the one who has the fewest first place votes among the remaining (non-eliminated) candidates. While it seems at first glance that the positional ballot scheme might be appropriate, it is difficult with the positional scheme to figure out who the ballot goes to if the candidate in the first position has been eliminated. A ballot with an eliminated candidate in the first position needs to be counted in the subsequent rounds as a vote for the candidate in the second position (or the third if the second candidate has also been eliminated, etc.).

Thus, we use the pairwise ballot scheme for our instant runoff voting manipulation reduction. We will also use the elimination mechanism described in Section 3.3.4. The determination of which candidates to eliminate in a round uses a count of first place votes, ignoring those candidates already eliminated. Since the set of eliminated candidates changes every round, the assignment of points to the remaining candidates also changes. Let us define a generalized subformula which will be true iff the candidate c has earned a

point from a voter v in round r. The points each candidate earns will later be summed and compared to determine who is eliminated for round $r+1$.

First, for a candidate c to earn any points, they must not already be eliminated.

$$\neg E_{r,c}$$

And second, they must be ranked higher on voter v's ballot than every other non-eliminated candidate d, which we may alternatively state as either d was already eliminated, or c must be ranked higher than d.

$$\bigwedge_{d \in C \setminus c} E_{r,d} \vee P_{v,c,d}$$

For future convenience, we embed each subformula for a first place point as the surrogate

$$\text{Point}_{c,v,r} \leftrightarrow \neg E_{r,c} \wedge \bigwedge_{d \in C \setminus c} E_{r,d} \vee P_{v,c,d}$$

To compute a candidate's fate in the next round, we compare their total number of first place points to that of each other candidate. The candidate with the fewest first place points in a given round is eliminated. If two candidates tie for fewest first place points, both are eliminated. This is equivalent to stating that candidates who have more points than some other non-eliminated candidate are protected from elimination, and those that fail to meet this criterion are eliminated.

We can tally the points of the each of the candidates as a score and then compare them with inequalities. Let

$$\text{Score}_{r,c} = \sum_{v \in V \cup M} \text{Point}_{c,v,r}$$

be the summation of the first place points c has earned in round r. For a particular round r, candidate c has more points than candidate d if and only if neither candidate has been eliminated and c's score is greater than d's score.

We then proceed as before and embed the status of the contest between each pair of candidates as a surrogate $B_{c,d,r}$ (B for "beats").

$$B_{c,d,r} \leftrightarrow \neg E_{r,c} \wedge \neg E_{r,d} \wedge (\text{Score}_{d,v,r} - \text{Score}_{c,v,r} \leq -1)$$

We should now be able to decide the elimination statuses for the next round. Candidates that "beat" at least one other candidate are protected from elimination and the others are eliminated. There is one issue that must be addressed however. Since we are using the "ties eliminate" model here, there is the possibility that more than one candidate could be eliminated per round. Since we fix the number of rounds at m, it may occur that there is exactly one candidate remaining before the last round. Obviously this candidate is the winner, but if the remaining rounds are carried out, under the current rules that last candidate will fail to beat any other candidate, simply because there are no other candidates to beat, and would thus be eliminated. We can correct this situation by adding as a requirement to elimination that there is some other candidate remaining in the race.

$$E_{r+1,c} \leftrightarrow \left(\left(\bigvee_{d \in C \setminus c} \neg E_{r,d} \right) \wedge \bigwedge_{d \in C \setminus c} \neg B_{c,d,r} \right)$$

All of the above clauses are instantiated for each round in $\mathcal{R} = \{0, \ldots, m-1\}$.

The final requirement is to make sure the preferred candidate c_p is the unique winner, which they are if they remain in round $m-1$ while everyone else is eliminated.

$$\neg E_{m-1,c_p} \wedge \bigwedge_{d \in C \setminus c_p} E_{m-1,d}$$

Plurality with two-candidate runoff

Plurality with two-candidate runoff shares the same basic mechanics as instant runoff voting: The only major change is the number of rounds and the criteria for elimination. Instant runoff voting has m rounds ($m-1$ elimination steps), and plurality with two-candidate runoff is fixed to three rounds

(two elimination steps). The criterion for passing from the first round to the second is more strict. As we have defined plurality with two-candidate runoff, if more than two candidates are tied in the first round they all proceed to the runoff round. If one candidate holds the lead by itself with others tied immediately behind it, all candidates in the pack that immediately trails are eliminated.

We have already defined most of the components needed to reduce manipulation of plurality with two-candidate runoff in Section 3.3.4 and in the reduction from manipulation of instant runoff voting. We will use the same representation for the ballots, and the same constraints on manipulator votes. The elimination mechanic will also be used. The set of rounds is simply $\mathcal{R} = 0, 1, 2$. It starts in the same manner,

$$\bigwedge_{c \in C} \neg E_{0,c}$$

There is now only one round for which elimination status must be carried over, we can specify that with simply

$$\bigwedge_{c \in C} \neg E_{1,c} \vee E_{2,c}$$

Observe that the above constraint is equivalent to the corresponding constraint from the reduction of instant runoff voting, but it is simplified because of the fixed number of rounds.

Next, we copy and use the points and pairwise outscore ("beats") mechanisms verbatim. In the next formula, we make use of the $B_{c,d,r}$ variables, and assume that they have been set as described in the reduction for instant runoff voting above. To advance, a candidate must either trail no other candidates, or trail exactly one and tie none. We introduce the variable

$$T_c^0 \leftrightarrow \bigwedge_{d \in C \setminus c} B_{d,c,0}$$

to represent candidate c trailing no other candidates. We also introduce

$$T_c^1 \leftrightarrow \neg T_c^0 \wedge \sum_{d \in C \setminus c} B_{d,c,0} \leq 1$$

to represent candidate c trailing exactly one other candidate. The candidate will advance if it either trails no one, or trails exactly one and ties none. The latter condition can be restated as "trails exactly one and beats everyone except himself and the one he trails." Thus we can simply count the number of candidates beaten and ensure it is at least $m - 2$.

Assembling all the above, the formula for elimination in round 1 for a candidate c (we give the contrapositive for clarity) is then

$$\neg E_{1,c} \leftrightarrow T_c^0 \vee \left(T_c^1 \wedge \sum_{d \in C \setminus c} B_{c,d,0} \geq m - 2 \right)$$

To advance to round 2, a candidate may lose to no other candidates in round 1.

$$E_{2,c} \leftrightarrow \bigwedge_{d \in C \setminus c} \neg B_{d,c,1}$$

And again, as with instant runoff voting, the requirement that the preferred candidate c_p is the unique winner is expressed by requiring that c_p remain, and everyone but c_p be eliminated in round 2.

$$\neg E_{2,c_p} \wedge \bigwedge_{d \in C \setminus c_p} E_{2,d}$$

Copeland voting

Copeland voting is one of the most basic possible extensions of the Condorcet criterion into a total voting system that always produces a winner. The Condorcet criterion says the winner should be the candidate that defeats all others in pairwise elections, if such a candidate exists. When one does not exist, Copeland voting tallies the number of pairwise elections that each candidate has won, and in the most common formulation awards one-half of

a point for a tie—but the prize for a tie could be anywhere in the range $[0, 1]$. Copeland manipulation is NP-complete for rational values other than $\frac{1}{2}$.

Because Copeland voting is based on pairwise rankings between two candidates on each ballot, this reduction also uses the pairwise representation of the ballot used above in instant runoff voting and plurality with two-candidate runoff. Copeland voting does not need the elimination mechanism, as there is only a single tally of the Copeland score. The Copeland score is calculated by determining the winner of each pairwise election between two candidates, and summing the points awarded to a candidate for winning or tying in each of its pairwise elections.

The following subformula computes the pairwise election results between two candidates. We reuse the letter B for "beats," but it is distinct from how it is defined and used in the previous reductions above. Here, $B_{c,d}$ is true iff there are more ballots that list candidate c is in a higher position than candidate d than list them in the other order.

$$B_{c,d} \leftrightarrow \sum_{v \in V \cup M} -1 \cdot P_{v,c,d} + 1 \cdot P_{v,d,c} \leq -1$$

There are three possible outcomes for a pairwise election: a victory for one candidate or the other, or a tie. This cannot be represented in a single variable. $B_{c,d}$ is distinct from $B_{d,c}$ because of the possibility of a tie in c and d's pairwise contest. One of these variables being true and the other false indicates that one candidate was victorious in the pairwise election, while both false indicates a tie. Both cannot be true simultaneously.

The Copeland scores of the candidates are calculated from the status of the "beats" variables. Every time the candidate achieves a pairwise win, they earn a single point. This leads to a problem with points for ties however. Since we are working only with integral value, fractional values cannot be used. To incorporate rational values for a tie into the Copeland voting reduction, point values are multiplied by the denominator of the tie value. In the case of Copeland when the purse awarded for a tie is $\frac{1}{2}$, the effective point values awarded by the reduction are 0 for a loss, 1 for a tie, and 2 for a win. If $\frac{2}{3}$ of a point are awarded for a tie, the values are 0, 2, and 3.

To determine the winner of the election, we check if there is a candidate that has a better Copeland score than the others. Summing the candidates' Copeland scores is done as described in Section 3.3.4. w_w, w_t are the point values associated with a pairwise win and tie respectively.

$$\text{Score}_c = \sum_{d \in C \setminus c} w_w \cdot (B_{c,d} \wedge \neg B_{d,c}) + \sum_{d \in C \setminus c} w_t \cdot (\neg B_{c,d} \wedge \neg B_{d,c})$$

To make the preferred candidate c_p the unique winner, c_p's Copeland score must be greater than that of every other candidate

$$\bigwedge_{d \in C \setminus c_p} \text{Score}_d - \text{Score}_{c_p} \leq -1$$

Chapter 4

Results

We used our software to compute the manipulation numbers of randomly generated elections. We had 48 test sets of elections, ranging in size from 1 to 128 non-manipulating voters in powers of two, having three and five candidates, and having votes distributed according to the uniform distribution, the Condorcet distribution with $p = 0.6$, and the spatial distribution using three dimensions. Each of these test sets contained 1000 elections. We used the FIND-FIRST method described in Section 3.1 to compute lower and upper bounds on each manipulation number for all 1000 elections in each data set, under the voting rules plurality, plurality with two-candidate runoff, veto, Borda count, instant runoff voting, and Copeland. Though we experimented using other solvers, all of our final results were computed by the Clasp solver solving CNF-SAT formulas [GKS10]. We did not use the pseudo-Boolean optimization capabilities of the Clasp solver because our reductions were crafted to target 0–1 integer programming and SATISFIABILITY. We look forward to adapting our reductions to target the pseudo-Boolean optimization problem in the future. For each instance of a manipulation problem (C, V, R, c_p, k), the solver was given two minutes to solve the reduction of the instance. With this time limit in place, the FIND-FIRST search procedure produced lower and upper bounds on each manipulation number of each instance.

4.1 Precision

Using a fixed two-minute time limit for each instance meant that some problems, especially those instances with more candidates and more non-manipulating voters, did not have their manipulation numbers computed exactly, but rather had them bounded by lower and upper bound estimates of their true value. We found that we were able to compute the manipulation numbers of most instances exactly, and those that we were not able to compute exactly we were able to bound reasonably tightly.

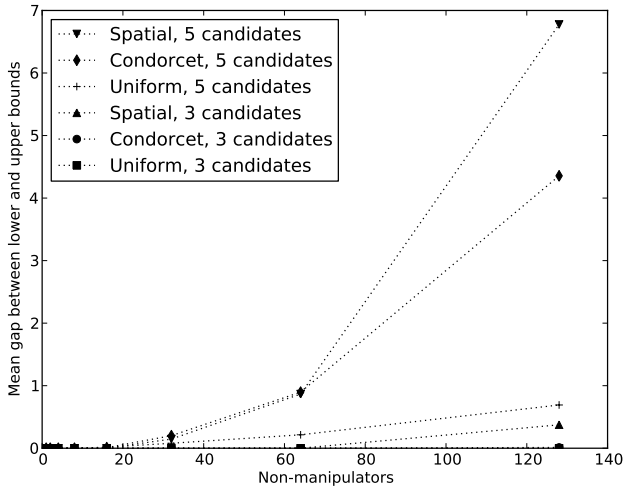

Figure 4.1: Mean size of the gap between the lower and upper bounds for all manipulation numbers and all rules.

Figure 4.1 shows the mean size of the gap between the lower and upper bounds we calculated for the manipulation numbers for each of the uniform, Condorcet, and spatial distributions, with three and five candidates. Figure 4.1 includes data for all rules and manipulation numbers. The greatest mean gap size is 6.77 for the spatial distribution with five candidates, and this occurs at 128 non-manipulators. Using our method, it is expected that one can compute the manipulation number for a random spatially distributed election on five or fewer candidates and 128 non-manipulators to

within 7 manipulators.

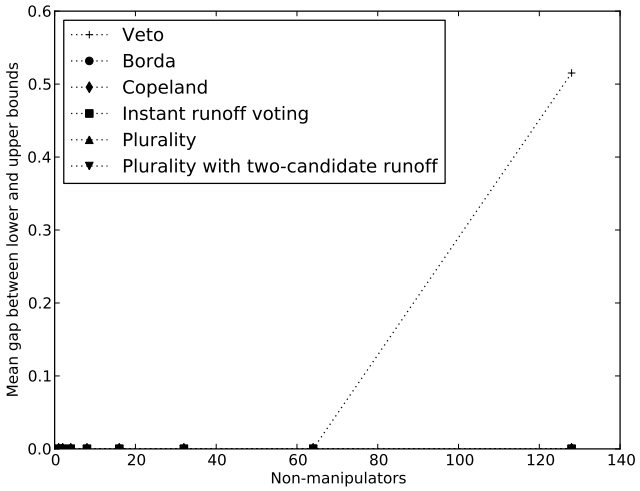

Figure 4.2: Mean size of the gap between the lower and upper bounds for all manipulation numbers and all distributions, with three candidates.

Figure 4.2 shows the mean size of the gap between the lower and upper bounds that the solver established on manipulation numbers it was computing. The value of zero for most of the data points means that for the vast majority of these small instances the solver was able to compute the exact value of the all manipulation numbers.

Only the veto rule presented any difficulty at this problem size. It is worth noting that the reduction for the veto rule was simply a naïve application of the general scoring protocol reduction using the scoring vector $\alpha = \langle 1, 1, 0 \rangle$. The solver had no special knowledge that only vetoes needed to be distributed amongst the non-preferred candidates. Instead, the solver was forced to attempt to arrange all the candidates on every ballot, trying to find some arrangement that would yield a better outcome. The lesson here is that generalized reductions can easily obscure meaningful structure in a problem that a solver could take advantage of if that structure were more apparent in the formula.

Figure 4.3 shows the corresponding data for five candidate elections. As

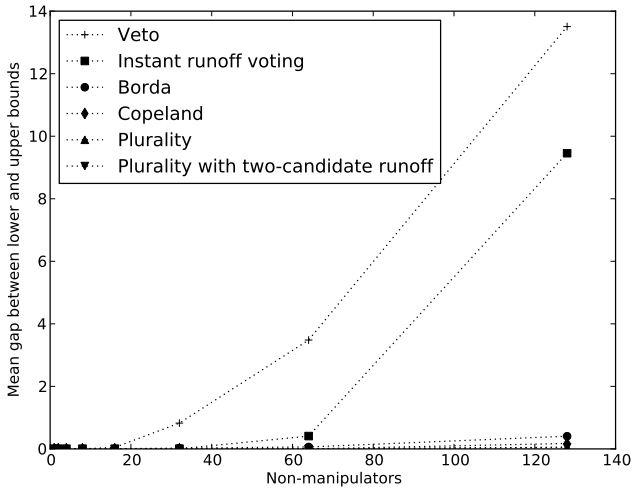

Figure 4.3: Mean size of the gap between the lower and upper bounds for all manipulation numbers and all distributions, with five candidates.

in Figure 4.2, we see the same issue with the veto rule. It is not surprising to see the same difficulty remain in larger problem instances. The problem is exacerbated with more candidates, further underscoring the need for a specialized reduction when additional structure is available.

Additionally we see some difficulty with instant runoff voting on the five candidate problems. This too is not particularly surprising, as instant runoff voting is the rule with the most complex reduction we examined in this work. Every round of eliminations is simulated in the formula. The result of the election is very far removed from the state of the ballots in the formulas, through a long chain of rounds and eliminations, and each elimination changes the interpretation of the ballots in the later rounds. Changes in the ballots likely entail a considerable avalanche of changes, giving deductions and learned conflict clauses minimal value.

4.2 Expected and actual manipulation numbers

The next measurement of interest is the actual size of the manipulation numbers of our test elections compared with their theoretical growth rates. Figure 4.4 shows the mean of all manipulation numbers except the first, or equivalently, the minimum coalition size to make a random candidate who would lose without the help of the manipulators become a winner. Immediately apparent is a large separation in the constant factor between the uniform, Condorcet, and spatial distributions, with uniform requiring a smaller coalition than Condorcet, and Condorcet in turn needing fewer manipulators than the spatial distribution. Not immediately apparent from the figure however is the growth rate of these minimum coalition sizes. Based on the analyses of Procaccia and Rosenschein, Xia and Conitzer, and Walsh, we would expect these minimum coalition sizes to grow as $\Theta(\sqrt{n})$ [PR07, XC08, Wal09b]. Figure 4.5 is a similar plot of manipulation numbers, but here they are divided by $\sqrt{n}$. While the observations for the uniform distribution appear in line with the theoretical expectation, the data for the Condorcet and spatial distributions are less well-behaved.

Examining the same progression of actual minimum coalition size grouping by the voting rule used, Figure 4.6 reveals that Borda appears to require the fewest manipulators to alter its outcomes. The relationship between the other voting rules is less distinct; there is no one rule clearly needing more manipulators.

Applying the same transformation as before and dividing the mean manipulation numbers by $\sqrt{n}$, Figure 4.7 shows that the mean manipulation numbers appear to be growing faster than the theoretical $\Theta(\sqrt{n})$ bound. This figure illustrates that this unexpected pattern of growth does not appear to be related to the voting rule. The apparent $\omega(\sqrt{n})$ growth appears to be related to the spatial and Condorcet distributions. The votes of the elections generated according to these distributions were independent and identically distributed. Procaccia and Rosenschein demonstrated that the Condorcet distribution satisfies the final condition required for their theorem to apply, and the spatial distribution also seems to satisfy it.

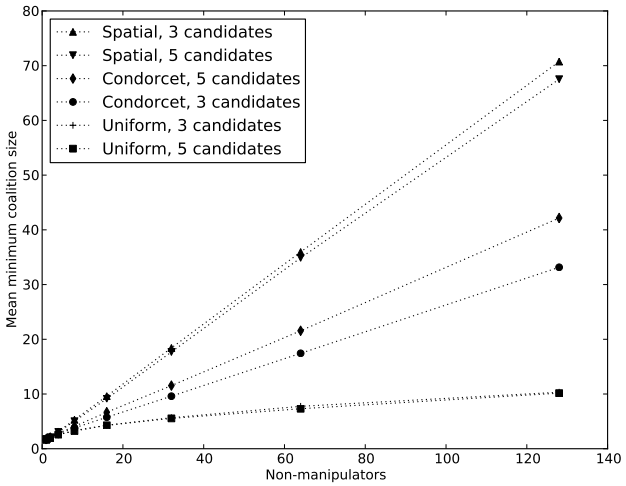

Figure 4.4: Mean minimum coalition size to make a random non-winning candidate win, for all rules.

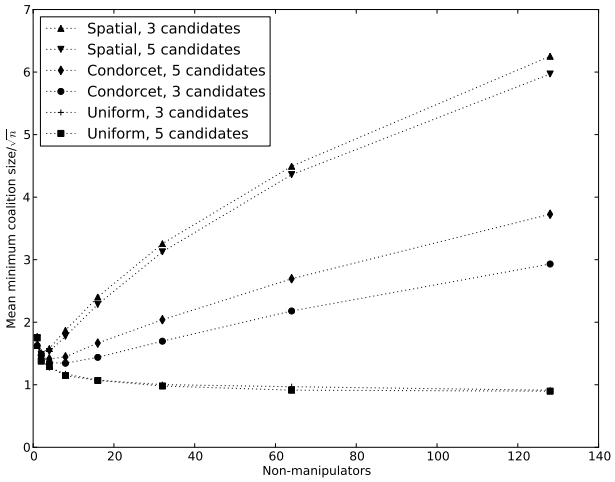

Figure 4.5: Mean minimum coalition size to make a random non-winning candidate win divided by the square root of the number of non-manipulators, for all rules.

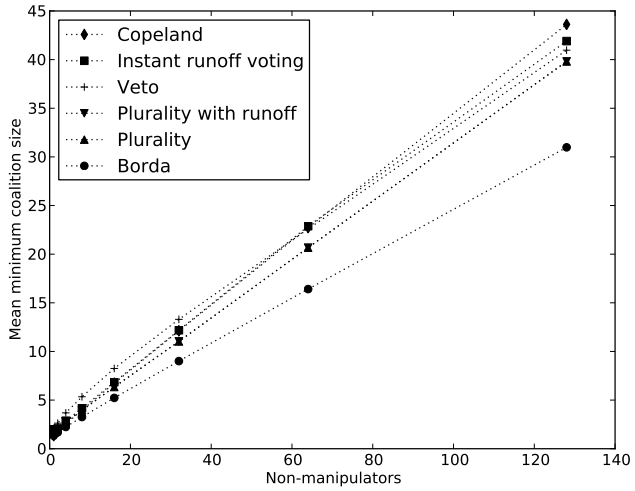

Figure 4.6: Mean minimum coalition size to make a random non-winning candidate win, for all distributions and three and five candidates.

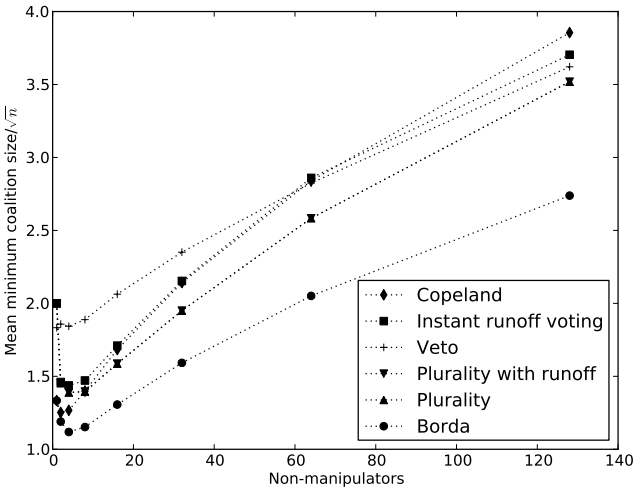

Figure 4.7: Mean minimum coalition size to make a random non-winning candidate win divided by the square root of the number of non-manipulators, for all distributions and three and five candidates.

Figure 4.8 is similar to Figure 4.7, but is restricted to instances from the uniform distribution. Here the growth rate appears to be well bounded by the theoretical $\Theta(\sqrt{n})$. Veto is seen to require the largest coalition on average, followed by Copeland, instant runoff voting, plurality and plurality with runoff, and Borda.

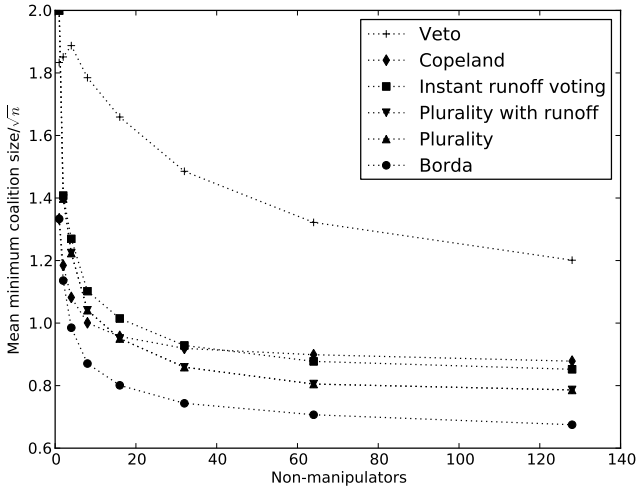

Figure 4.8: Mean minimum coalition size to make a random non-winning candidate win divided by the square root of the number of non-manipulators, for the uniform distribution only and three and five candidates.

4.3 Two-to-three possible winner gap

Conitzer and Sandholm argued for the non-existence of voting rules that are usually hard to manipulate [CS06] by considering the case where two candidates could be made to win, but three could not. They presented the procedure Find-Two-Winners which, given an instance of a manipulation problem where exactly two candidates could be made to win, it would identify those candidates and provide the votes for the manipulators to cast. They argued that the manipulation problem of voting rules that are desirable to use in practice would have an overwhelming proportion of instances

for which FIND-TWO-WINNERS is applicable. Figure 4.9 shows the mean of the second and third manipulation numbers—the minimum coalition sizes needed to make two candidates and three candidates win—and the regions where FIND-TWO-WINNERS works. We can see that the window where we can trust the output of FIND-TWO-WINNERS is relatively small across all the distributions we tested.

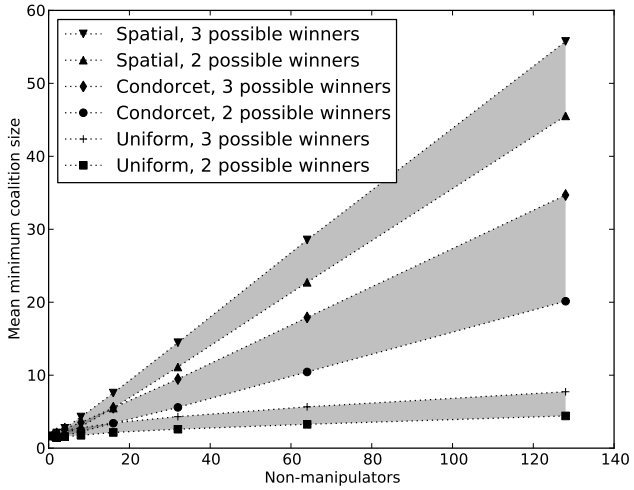

Figure 4.9: Mean minimum coalition size to make two or three candidates out of five possible unique winners, for all voting rules.

The same pattern appears when data is grouped by the voting rule. Figure 4.10 illustrates a similar occurrence with the Copeland, instant runoff voting, and Borda rules. Figure 4.11 shows the same for the plurality, plurality with two-candidate runoff, and veto rules.

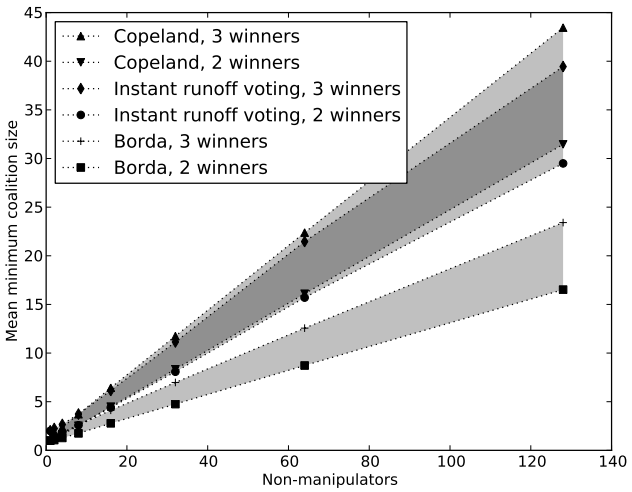

Figure 4.10: Mean minimum coalition size to make two or three candidates out of five possible unique winners, for all distributions.

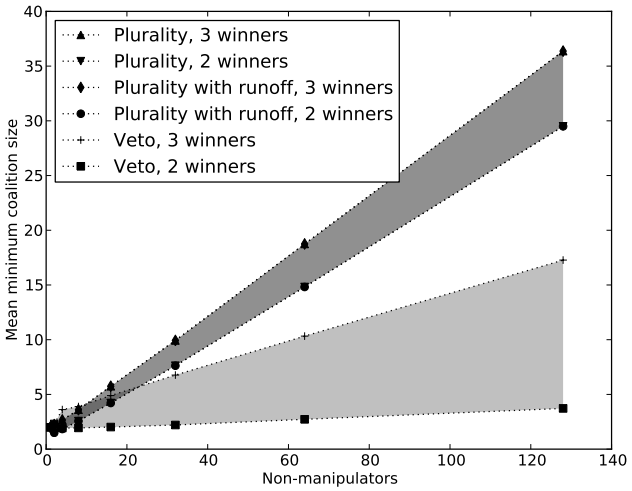

Figure 4.11: Mean minimum coalition size to make two or three candidates out of five possible unique winners, for all distributions.

Chapter 5

Conclusions

By reducing manipulation problems to SATISFIABILITY and employing state-of-the-art solvers we were able to investigate the manipulation power of coalitions in a variety of circumstances, and to determine the minimum number of manipulators a coalition would need to be able to manipulate an election. By computing manipulation numbers using reductions it was a straightforward job to produce results that we were confident were correct. The reductions were essentially literal simulations of the various voting rules that tasked the solver with working backwards from a foregone conclusion (that a preferred candidate is the unique winner of an election) through to a set of manipulator ballots that accomplished the assumed result. By using an appropriate search harness, we were able to construct very precise estimates of the manipulation numbers for the test elections, and where we were not able to determine the exact value of a manipulation number, we were able to establish lower and upper bounds. Though the largest instances tested (those with five candidates and 128 non-manipulators) had larger gaps between the bounds, this is because of the fixed time-limit permitted to each individual solver instance. Raising this limit would surely result in more precise bounds.

The simplicity of the reductions was both an aid and a hindrance. We were able to construct reductions simply and with great confidence that they were correct, but their simplicity was also their undoing. As was seen

in the performance the reduction for the veto rule, the solver had difficulty solving what should have been an easy problem. The implementation of the veto reduction using a generalized reduction for scoring rules was clearly suboptimal. Where an optimal algorithm would only have to decide which candidates to veto (and in the unweighted case it becomes a matter of simple arithmetic), the solver attempted to find solutions by searching through the entire state space of the manipulators' ballots. We have seen that adding redundant clauses to a formula can result in dramatic gains in solver performance [Heu09]. Clearly there is value in attempting to provide more information and reveal the essential structure of the problem to the solver in any reduction, as our difficulty with the veto rule illustrates. Future work would do well to consider more carefully the actual techniques and strategies employed by state-of-the-art solvers, and design reductions that simplify the solver's task and maximize its efficiency.

The pseudo-Boolean optimization problem and its associated state-of-the-art solvers presents further prospects for future research. The definition of pseudo-Boolean optimization seems to be a very natural reduction target for manipulation problems. It is very similar to the combination of 0–1 integer programming and CNF-SAT that we used in this work, allowing 0–1 integer programming-style inequalities alongside CNF-SAT clauses. pseudo-Boolean optimization also has an annual competition for solvers that occurs as part of the annual Conference on Theory and Applications of Satisfiability Testing [MR10, SS10].

In our results, we could see that the manipulation numbers of elections drawn from the uniform distribution were consistent with the results of Procaccia and Rosenschein in that their growth was bounded as $\Theta(\sqrt{n})$. Our figures for the Condorcet and spatial distributions however did not show the same consistency, despite both distributions meeting the conditions for Procaccia and Rosenschein's theorem to apply. It is likely that we did not analyze elections with sufficient numbers of non-manipulators for the $\Theta(\sqrt{n})$-bounded growth to appear. With continuing advances in the state-of-the-art in SATISFIABILITY or pseudo-Boolean optimization solvers, combined with improved reductions that minimize wasted work and max-

imize the applicability of learned conflict clauses, this technique could be extended to work for even larger instances.

We could also see that the region where manipulation problems are solvable by Conitzer and Sandholm's FIND-TWO-WINNERS is not as large as previously thought. FIND-TWO-WINNERS is a useful algorithm to apply if one can be confident that a given coalition size is likely to be in the region of one to two possible winners rather than the region of two or more possible winners. We have seen due to Walsh that for the veto rule, uniformly distributed elections and a coalition that is $3\sqrt{n}$ in size is over 95% likely to be able to make a random candidate win [Wal09b]. Our results for other voting rules, also for the uniform distribution, show that the mean minimum coalition size to make a random non-winning candidate win tends to be between $0.6\sqrt{n}$ and $1.4\sqrt{n}$. These numbers could serve as a useful guideline for when a manipulating coalition could be confident in applying FIND-TWO-WINNERS, and when it should consider other methods.

Appendix A

Code

Our software implementation of the election test set generator, brute force manipulation number solver, manipulation problem reductions, common voting rule reduction constructs, FIND-FIRST and the associated search harness are all available from our `git` repository at `http://github.com/cconnett/nplib`.

Bibliography

[Arr51] Kenneth Arrow. *Social Choice and Individual Values*. Cowles Foundation, New Haven, first edition, 1951.

[Ber85] Sven Berg. Paradox of voting under an urn model: The effect of homogeneity. *Public Choice*, 47:377–387, 1985.

[BHZ06] Lucas Bordeaux, Youssef Hamadi, and Lintao Zhang. Propositional satisfiability and constraint programming: A comparative survey. *ACM Computing Surveys*, 38(4):12, 2006.

[Bla48] Duncan Black. On the rationale of group decision-making. *Journal of Political Economy*, 56(1):23–34, 1948.

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

[BO91] John Bartholdi, III and James Orlin. Single transferable vote resists strategic voting. *Social Choice and Welfare*, 8(4):341–354, 1991.

[BRS09] Daniel Le Berre, Olivier Roussel, and Laurent Simon. SAT competitions, May 2009. `http://www.satcompetition.org/`.

[BTT89a] John Bartholdi, III, Craig Tovey, and Michael Trick. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6(3):227–241, 1989.

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

[CC78] John Chamberlin and Michael Cohen. Toward applicable social choice theory: A comparison of social choice functions under spatial model assumptions. *The American Political Science Review*, 72:1341–1356, 1978.

[Cha85] John Chamberlin. Investigation into the relative manipulability of four voting sytems. *Behavioral Science*, 30:195–203, 1985.

[Con06] Vincent Conitzer. *Computational Aspects of Preference Aggregation*. PhD thesis, Carnegie Mellon University, 2006.

[Coo71] Stephen Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, New York, NY, USA, 1971. ACM.

[CS02] Vincent Conitzer and Tuomas Sandholm. Complexity of manipulating elections with few candidates. In *Proceedings of the Eighteenth National Conference on Artificial intelligence*, pages 314–319, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[CS03] Vincent Conitzer and Tuomas Sandholm. Universal voting protocol tweaks to make manipulation hard. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 781–788, San Francisco, CA, USA, August 2003. Morgan Kaufmann Publishers Inc.

[CS06] Vincent Conitzer and Tuomas Sandholm. Nonexistence of voting rules that are usually hard to manipulate. In *Proceedings of the Twenty-first National Conference on Artificial Intelligence*, pages 627–634. AAAI Press, July 2006.

[DHO70] Otto Davis, Melvin Hinich, and Peter Ordeshook. An expository development of a mathematical model of the electoral process. *American Political Science Review*, 54(2):426–448, 1970.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[DS00] John Duggan and Thomas Schwartz. Strategic manipulability without resoluteness or shared beliefs: Gibbard-Satterthwaite generalized. *Social Choice and Welfare*, 17:85–93, 2000.

[EL05] Edith Elkind and Helger Lipmaa. Small coalitions cannot manipulate voting. In *The Commonwealth of Dominica*, pages 285–297. Springer-Verlag, 2005.

[ER91] Eithan Ephrati and Jeffrey Rosenschein. The Clarke tax as a consensus mechanism among automated agents. In *Proceedings of the National Conference on Artificial Intelligence*, pages 173–178, San Jose, California, July 1991.

[ER93] Eithan Ephrati and Jeffrey Rosenschein. Multi-agent planning as a dynamic search for social consensus. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 423–429, Chambéry, France, 1993.

[FHHR09] Piotr Faliszewski, Edith Hemaspaandra, Lane Hemaspaandra, and Jörg Rothe. The shield that never was: Societies with single-peaked preferences are more open to manipulation and control. In *Proceedings of the Twelfth Converence on Theoretical Aspects of Rationality and Knowledge*, pages 118–127, New York, NY, USA, 2009. ACM.

[FHS08] Piotr Faliszewski, Edith Hemaspaandra, and Henning Schnoor. Copeland voting: Ties matter. In *Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems*, pages 983–990, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[FHS10] Piotr Faliszewski, Edith Hemaspaandra, and Henning Schnoor. Manipulation of Copeland elections. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems*, Toronto, Ontario, Canada, 2010. To appear.

[FKN08] Ehud Friedgut, Gil Kalai, and Noam Nisan. Elections can be manipulated often. In *Proceedings of the Forty-ninth Annual IEEE Symposium on Foundations of Computer Science*, pages 243–249, Washington, DC, USA, 2008. IEEE Computer Society.

[Gib73] Allan Gibbard. Manipulation of voting schemes. *Econometrica*, 41:587–602, 1973.

[GKS10] Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Clasp: A conflict-driven nogood learning answer set solver, May 2010. `http://www.cs.uni-potsdam.de/clasp/`.

[Heu09] Marijn Heule. Solving edge-matching problems with satisfiability solvers. In *SAT 2009 Competitive Events Booklet: preliminary version*, pages 69–82, 2009.

[HH06] Christopher Homan and Lane Hemaspaandra. Guarantees for the success frequency of an algorithm for finding Dodgson-election winners. In *Proceedings of the Thirty-first International Symposium on Mathematical Foundations of Computer Science*, pages 528–539. Springer-Verlag, 2006.

[IKM10] Marcus Isaksson, Guy Kindler, and Elchanan Mossel. The geometry of manipulation — a quantitative proof of the Gibbard-Satterthwaite theorem. Manuscript, April 2010.

[Kar72] Richard Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[Kre98] Keith Krehbiel. *Pivotal Politics: A Theory of U.S. Lawmaking.* University of Chicago Press, 1998.

[KS98] Donald Kreher and Douglas Stinson. *Combinatorial Algorithms: Generation, Enumeration, and Search.* CRC Press LLC, Boca Raton, Florida, 1998.

[Lev84] Leonid Levin. Problems, complete in "average" instance. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, page 465, New York, NY, USA, 1984. ACM.

[LZD04] Ruiming Li, Dian Zhou, and Donglei Du. Satisfiability and integer programming as complementary tools. In *Proceedings of the 2004 Asia and South Pacific Design Automation Conference*, pages 879–882, 2004.

[Mak] Andrew Makhorin. GNU Linear Programming Kit. `http://www.gnu.org/software/glpk/`.

[MMSOS98] Vasco Manquinho, João Marques-Silva, Arlindo Oliveira, and Karem Sakallah. Satisfiability-based algorithms for 0-1 integer programming. Technical report, Cadence European Laboratories / INESC, Instituto Superior Técnico, 1000 Lisboa, Portugal, 1998.

[MR10] Vasco Manquinho and Olivier Roussel. Pseudo-boolean competition 2010, May 2010. `http://www.cril.univ-artois.fr/PB10/`.

[Mur83] Katta Murty. *Linear Programming.* Wiley, 1983.

[NW87] Richard Niemi and John Wright. Voting cycles and the structure of individual preferences. *Economic Theory*, 7(3):173–183, 1987.

[Pap81] Christos Papadimitriou. On the complexity of integer programming. *Journal of the ACM*, 28(4):765–768, 1981.

[PR97] Keith Poole and Howard Rosenthal. *Congress: A Political-Economic History of Roll Call Voting*. Oxford University Press, 1997.

[PR07] Ariel Procaccia and Jeffrey Rosenschein. Average-case tractability of manipulation in voting via the fraction of manipulators. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 718–720, New York, NY, USA, 2007. ACM.

[Pro10] Ariel Procaccia. Can approximation circumvent Gibbard-Satterthwaite? In *Proceedings of the Twenty-fourth AAAI Conference on Artificial Intelligence*, July 2010. To appear.

[Sat75] Mark Satterthwaithe. Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10:187–217, 1975.

[Sch03] Marcus Schulze. A new monotonic and clone-independent single-winner election method. *Voting Matters*, 17:9–19, October 2003.

[SP08] Carsten Sinz and Hendrik Post. SAT-Race 2008, June 2008. `http://baldur.iti.uka.de/sat-race-2008/`.

[SS10] Ofer Strichman and Stefan Szeider. SAT2010 — Thirteenth international conference on theory and applications of satisfiability testing, July 2010. `http://ie.technion.ac.il/SAT10/`.

[Tse68] Gregory Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic*, pages 115–125, 1968.

[TW01] Wade Trappe and Lawrence Washington. *Introduction to Cryptography with Coding Theory.* Prentice-Hall, Upper Saddle River, NJ, 2001.

[Wal09a] Toby Walsh. Manipulability of single transferable vote. In *Proceedings of the CARE'09 International Workshop on Collaborative Agents — REsearch and Development*, 2009.

[Wal09b] Toby Walsh. Where are the really hard manipulation problems? The manipulation phase transition. In *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence*, pages 324–329, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[War98] Joost Warners. A linear-time transformation of linear inequalities into conjunctive normal form. *Information Processing Letters*, 68(2):63–69, 1998.

[WN99] Laurence Wolsey and George Nemhauser. *Integer and Combinatorial Optimization.* Wiley, 1999.

[Woo94] Douglas Woodall. Properties of preferential election rules. *Voting Matters*, 3, December 1994. `http://www.mcdougall.org.uk/VM/ISSUE3/P5.HTM`.

[XC08] Lirong Xia and Vincent Conitzer. A sufficient condition for voting rules to be frequently manipulable. In *Proceedings of the Ninth ACM Conference on Electronic Commerce*, pages 99–108, New York, NY, USA, 2008. ACM.