Bribery and Voter Control under Voting-Rule Uncertainty[*]

Gabor Erdélyi
Universität Siegen
Siegen, Germany
erdelyi@wiwi.uni-siegen.de

Edith Hemaspaandra
Rochester Institute of Technology
Rochester, NY, USA
eh@cs.rit.edu

Lane A. Hemaspaandra
University of Rochester
Rochester, NY, USA
www.cs.rochester.edu/∼lane

ABSTRACT

We study manipulative actions, especially bribery and control, under "voting-rule uncertainty," which means that there are two or more rules that may be used to determine the election's outcome. Among our highlights are that we show a new case in which "ties matter," we link manipulation and bribery in a way that shows many cases of single-bribery to be in polynomial time, we explore the relations between the bribery and control complexities of the underlying systems and their "uncertain" combination, and we obtain many results about the complexity of natural voting rules under voting-rule uncertainty, most notably regarding control by adding voters under election families of the form $\{k_1\text{-Approval},\dots,k_\ell\text{-Approval}, \hat{k}_1\text{-Veto},\dots,\hat{k}_{\hat{\ell}}\text{-Veto}\}$.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Algorithms, Theory

Keywords

computational social choice, voting rules

1. INTRODUCTION, BACKGROUND, AND RELATED WORK

Voting rules and elections have been in use for a long time in many different areas such as political science, economics, and operations research. During the last few decades, voting has become more and more important to various areas of computer science, most notably in multiagent systems where voting has found uses in planning [9], similarity search [11], and the design of recommender systems [17] and ranking algorithms [7], just to name a few real-world examples. In many cases—especially in computer-science applications—the elections may be huge.

[*]Supported in part by grants DFG-ER-738/1-1 and NSF-CCF-{0915792,1101452,1101479}, and ESF COST Action IC1205. This work was done in part while G. Erdélyi was visiting the University of Rochester.

Appears in: *Alessio Lomuscio, Paul Scerri, Ana Bazzan, and Michael Huhns (eds.), Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), May 5-9, 2014, Paris, France.*

The study of the computational complexity of problems—such as manipulation and procedural control—related to voting rules was initiated by Bartholdi, Tovey, and Trick [2, 3]. Since manipulative attacks can now be conducted using great computational power, it is quite important to understand the computational complexity of manipulation problems. The present paper contributes to the study of several types of manipulative actions, such as manipulation [2], where a voter may cast an insincere vote, bribery [12], where an external agent may change some voters' votes, and control [3], where again an external agent—traditionally called the chair—may change the structure of an election in order to affect the outcome of the election.

Traditionally, the complexity of manipulative actions in elections is studied under the full information assumption, i.e., it is assumed that all relevant information is known to the manipulative agent(s). Recently, several papers have addressed models where some kind of uncertainty is introduced, e.g., [20, 1, 6, 4]. Our paper studies a model where there is uncertainty regarding the voting rule. As motivation regarding voting-rule uncertainty, we mention that we certainly have seen CS department chairs solicit preference-order votes from faculty members over hiring candidates, without informing the faculty of what rule will be used to select the winner(s). And one can easily imagine local or online elections in which it is known that k-approval will be used, but the exact value of k is decided only later.

In particular, this paper is in a model in which the manipulative agent does not know which voting rule will be used, but knows that it will be chosen from a family of voting rules, and the manipulative actor even while blind to which voting rule will be used wants to ensure that a given candidate will win. As far as we know, there is only one previous paper studying this model, namely the interesting paper of Elkind and Erdélyi [8] that introduced the model. (This model captures the voting rule being chosen after the manipulative action and adversarially. See, e.g., [4] for the collaborative case.)

The Elkind-Erdélyi paper studied only manipulation. In contrast, the present paper also studies bribery and, especially, control. For manipulation, we prove the first case within this model where "ties matter." For bribery and constructive control by adding voters, we show that every possible combination of easy/hard complexities can be realized (Table 1). Most importantly, as Classification 6.9 we broadly classify the constructive-control-by-adding-voters complexity of families of k-Approval and k-Veto rules under voting-rule uncertainty. The individual voting rules from this family, *without any uncertainty present*, were previously studied, most notably by Lin [22]. Our results study such families under voting-rule uncertainty, and range from polynomial-time algorithms established by interesting use of network-flow techniques and edge covering/matching algorithms to NP-hardness results based on set matching and covering. The differences between easiness and hardness might be hard to have guessed ahead of time, e.g., $\{$1-Approval, 1-Veto$\}$-CCAV is in P by a network-flow proof (that precisely captures the nature of this problem) and $\{$1-Veto, 2-Veto$\}$-CCAV is in P by an interesting use of edge cover, but $\{$2-Approval, 1-Veto$\}$-CCAV is NP-complete (even though 2-Approval-CCAV and 1-Veto-CCAV are each in P; uncertainty extracts a complexity-theoretic cost here). Network flows are a powerful technique and have been used in the study of manipulative actions on elections, e.g., [12, 5, 8]; surprisingly, the first such use was, to the best of our knowledge, quite recent [12].

2. PRELIMINARIES

An election is given by a pair $E = (C,V)$, where $C = \{c_1,\dots,c_m\}$ is a finite set of candidates and $V = \{v_1,\dots,v_n\}$ is a finite collection of voters. We'll represent preferences over candidate sets by tie-free linear orders, e.g., $a > b > c$ denotes a is strictly preferred to b who is strictly preferred to c. Each voter will have such a preference over the set of candidates. In our constructions, we sometimes as a shorthand will write a set in a preference order, in cases where the order within the set is irrelevant. For example, if we say a vote is $a > Z$, where $Z = \{b,c\}$, then it means that any vote in which a is preferred to b and c can be used in the construction; if the reader wants specificity, he or she can assume that such sets are written out using lexicographic order. We'll also sometimes use "$\cdots$" in a preference order, always in ways that will be clear from context. A voting rule is a function f that takes as its input an election, $E = (C,V)$, and outputs a set W, $W \subseteq C$. If $c \in W$, c is said to be a winner of the election E (under voting rule f).

A voting rule is said to be resolute if whenever there is at least one candidate there is exactly one winner. In the following we will specify some voting rules considered in this paper. (All these definitions are given for the case of unweighted votes. We refer to weighted voting regarding weighted types of bribery, and for all the systems below it is standard and self-apparent how to extend the notion to the weighted case.)

Condorcet A candidate c who is preferred to each other candidate by a strict majority of the voters is said to be a Condorcet winner. If there is no such candidate then there is no Condorcet winner.

Scoring Rules An (m-candidate) scoring rule is specified by a scoring vector $\alpha = (\alpha_1,\dots,\alpha_m)$ of integers $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_m \geq 0$. Each voter gives his or her ith-ranked candidate α_i points. The score of a candidate is the sum of points he or she gets from the voters. The candidate(s) with the maximum score are the winner(s) of the election. Typical scoring rules include m-candidate k-Approval, with scoring vector $\alpha = (\underbrace{1,\dots,1}_{k},\underbrace{0,\dots,0}_{m-k})$, and m-candidate k-Veto, with scoring vector $\alpha = (\underbrace{1,\dots,1}_{m-k},\underbrace{0,\dots,0}_{k})$, i.e., it is $(m-k)$-Approval. To build a scoring system that can handle any number of candidates, which is what a voting rule is supposed to do, families of scoring rules are used. The most important such families are defined as follows. For each k, k-Approval (k-Veto) is the election system that on inputs with m candidates employs the m-candidate k-Approval (k-Veto) system. 1-Approval is also known as Plurality. Let $score^k_V(c)$ denote the k-Approval score of candidate c under the voter set V and let $vetoes^k_V(c)$ be the number of k-vetoes that c receives from voters in V, i.e., the number of voters in V that rank c in one of the last k positions. Note that the k-Veto score of candidate c under the voter set V is exactly $\|V\| - vetoes^k_V(c)$.

Majority and MAJORITY In Majority voting, the candidate (if any) who is the top choice in the preference orders of a strict majority of the voters is the winner, and otherwise nobody wins. Elkind and Erdélyi [8] use a different notion of Majority, which we will distinguish from Majority by writing it in small caps, namely as MAJORITY. In MAJORITY voting, the candidate (if any) who is the top choice in the preference orders of a strict majority of the voters is the winner, and otherwise everyone wins. We'll discuss both notions in our manipulation section since our results there are making a point that intertwines with and contrasts with a result of Elkind and Erdélyi [8], and we'll use just Majority everywhere else.

Bucklin In Bucklin voting, for i equals one, then two, then..., we see if at least one candidate is among the first i choices of a strict majority of the voters. And at the first i for which there are any such candidates, the candidate(s) who occur within the first i places on the most votes are the winner(s).

Approval and ResoluteApproval These voting systems differ from what we have discussed so far, in that each voter votes not with a preference order but with a 0-1 vector indicating disapproval or approval of each candidate. The approval score of a candidate c is the number of voters who approve of c. The candidate(s) with the maximum approval score are the winner(s) of the election under Approval. In ResoluteApproval, the winner is the lexicographically largest among the Approval winners (and so of course there are never tied winners under ResoluteApproval). We will use the following artificial rules only in Table 1. ResoluteApproval$'$ is the voting rule where the winner of the election is the first candidate in the lexicographic ordering of all candidates if the ResoluteApproval winner is the last candidate in the lexicographic ordering, otherwise, the winner is the lexicographic successor of the winner under ResoluteApproval. In Approval$_{\text{even}}$, if no candidate is named dummy everyone loses. If there is a dummy candidate who has an even number of approvals, every candidate is a winner. Otherwise, we conduct Approval over all candidates except the dummy candidate. Approval$_{\text{odd}}$ is defined analogously with the difference that we check whether the dummy candidate has an odd number of approvals.

3. PROBLEM STATEMENT

Given a collection $\mathscr{F}$ of voting rules, we consider the complexity of manipulative actions—manipulation, bribery, and control—in elections where it is at the point of the manipulative action unknown which of the voting rules in $\mathscr{F}$ will be used, and the manipulative actor wants to make a given candidate win regardless of which system is ultimately employed. For manipulation this was defined by Elkind and Erdélyi [8], and the study in this model of bribery and control is new to this paper. Except in Section 4 when contrasting with Elkind and Erdélyi (who worked in the so-called unique-winner model, where one must make a given candidate the one and only winner), our entire paper is in the nonunique winner model, i.e., we speak of whether a candidate can be made "*a* winner." The nonunique-winner model avoids merging issues of tie-breaking with issues of winning, and so we find this the more natural model to use.

Manipulation under voting-rule uncertainty is formally defined as follows.

$\mathscr{F}$-SINGLE MANIPULATION ($\mathscr{F}$-SM)	
Given:	A candidate set C, a collection V_1 of voters with preferences over C, and a candidate $p \in C$.
Question:	Is there a vote v such that when v is added to V_1 p is a winner of the resulting election under each voting rule in $\mathscr{F}$?

Single manipulation is sometimes referred to simply as "manipulation" in the literature. Note that our above definition requires a vote v that succeeds under *all* the voting rules in $\mathscr{F}$—not separate votes v for each voting rule in $\mathscr{F}$. As usual, this paper treats the vote sets as coming in as a list of ballots.

Our second problem is bribery. Here, an external agent changes some voters' votes in order to reach his or her goal.

	$\mathscr{F}$-BRIBERY
Given:	An election $E = (C,V)$, a candidate $p \in C$, and a nonnegative integer k.
Question:	Is it possible to change up to k voters' votes in such a way that p is a winner of the resulting election under each voting rule in $\mathscr{F}$?

The above definition is of unweighted, unpriced bribery, which is the most basic case. However, we mention in passing that the examples in the bribery column of Table 1, which achieve all six possible easy/hard relationships, are chosen so as to also work for weighted bribery, priced bribery, and weighted priced bribery (as standardly defined [12]).

If we alter the above definition to omit k and simply treat $k = 1$ as the limit on the number of bribes, we call the resulting problem $\mathscr{F}$-SINGLE BRIBERY, or $\mathscr{F}$-SB for short.

There are many types of control specified in the literature. Hemaspaandra, Hemaspaandra, and Rothe [19] present the standard ones, and explain the motivations for each. However, in the present paper we focus solely on arguably the most important type of control, namely, constructive control by adding voters (CCAV, for short). In CCAV, the manipulative actor (the "chair") seeks to make his or her favorite candidate win the election by adding to the election some new voters from a set of unregistered voters. This is an (abstract) model that is inspired by such real-world activities as highly targeted phone banks, get-out-the-vote drives, offering senior citizens rides to the polling place, and so on. (The "$\cup$" in the definition below is of course a multiset union; different voters may vote in the same way.)

	$\mathscr{F}$-CCAV
Given:	An election $E = (C,V)$, a collection W of unregistered voters with preferences over C, a candidate $p \in C$, and a nonnegative integer k.
Question:	Is there a subcollection $W' \subseteq W$, $\|W'\| \leq k$, such that p is a winner of the election $(C, V \cup W')$ under each voting rule in $\mathscr{F}$?

4. MANIPULATION

This paper uses the nonunique-winner model (a.k.a. the co-winner model), which asks whether a given candidate can be made, or can be prevented from being, *a* winner of the election. As mentioned earlier, we prefer this model, as it avoids merging issues of tie-breaking with issues of winning. However, the literature also has papers studying the so-called unique-winner model, which asks about making a candidate be, or not be, *a sole winner* (i.e., having the candidate win and having no other candidate also win).

For a long time, these two models seemed to always give the same results, and papers—in retrospect perhaps somewhat naively—commented that tie-breaking details seemed not to matter. However, Faliszewski, Hemaspaandra, and Schnoor [14] then showed a context, regarding Copeland elections, in which these two models produced dramatically different complexity results (and see also the tie-related work in [18, Footnote 11] and [23]). Nonetheless, it remains the case that these two winner models have very, very few examples where they produce different complexities.

We provide a new case where the two models differ dramatically. One of the results of Elkind and Erdélyi [8] is that $\{$MAJORITY, STV$\}$-SM is in P, in the unique-winner model (which is the model of their paper). STV stands for single-transferable vote, and is an important election system (due to space, please see [8] for the definition).

However, we claim that in the nonunique-winner model, the same problem jumps up to being NP-complete. (The reason for this contrast is that $\{$MAJORITY, STV$\}$-SM is the same as MAJORITY-SM in the unique-winner model but is the same as STV-SM in the nonunique-winner model.)

THEOREM 4.1.

1. *[8]* $\{$MAJORITY, STV$\}$-SM *is in P in the unique-winner model.*
2. $\{$MAJORITY, STV$\}$-SM *is NP-complete (in the nonunique-winner model, which is our standard model in this paper).*

So this is a new case where "ties matter," i.e., the winner model makes a dramatic complexity difference (unless $P = NP$).

On the other hand, we note that in both the unique-winner and the nonunique-winner models, $\{$Majority, STV$\}$-SM coincides with Majority-SM, and so is in P, and so for "traditional" majority the above "ties matter" contrast disappears. Perhaps one should also say that this is a case where "(the definition of) Majority matters"!

5. BRIBERY

Elkind and Erdélyi [8] showed that for manipulation (in their paper's model), when combining two rules any combination of easy or hard rules can yield easiness or hardness under voting-rule uncertainty. Can we establish the same for bribery? Yes; Theorem 5.1 below states that for bribery every combination of easiness and hardness can be realized when one combines two rules under voting-rule uncertainty.

THEOREM 5.1. *For bribery of each of the four standard types (unpriced, unweighted bribery; priced, unweighted bribery; unpriced, weighted bribery; and priced, weighted bribery), for each of the eight cases (six cases if one ignore symmetric duplicates) as to whether f_1 bribery is in P or is NP-complete, whether f_2 bribery is in P or is NP-complete, and whether $\{f_1, f_2\}$ bribery is in P or is NP-complete, there are rules f_1 and f_2 realizing the given case.*

(The theorem is certainly not asserting that for all rules the bribery complexity is in P or NP-complete. Rather, it is exploring what can be realized using rules from these two most important complexity classes.) Due to space and since our main focus is on control, where we study in detail natural systems, we omit the proof of this theorem, although in Table 1 we provide examples of systems that realize each of the cases (the systems mentioned in the table are chosen to each establish their table line's result for all four types of bribery simultaneously). (Regarding the table, the voting rule AllWinner sets $W = C$ and the voting rule NoWinner sets $W = \emptyset$.)

Single bribery is in effect doing a logical "or" over a number of 1-manipulator manipulation problems. This was noted by [12], and we note that this holds even in the case of voting-rule uncertainty. (The $\leq_{dtt}^{p}$ below is just a technical way of capturing this; it denotes polynomial-time disjunctive truth-table reductions [21].)

PROPOSITION 5.2. *For each $\mathscr{F}$, $\mathscr{F}$-SB $\leq_{dtt}^{p}$ $\mathscr{F}$-SM.*

This connection lets us carry results for manipulation over to results for bribery.

	BRIBERY	CONSTRUCTIVE CONTROL BY ADDING VOTERS
easy + easy = easy	{AllWinner, NoWinner}	{1-Approval, 1-Veto} [Thm. 6.2]
easy + easy = hard	{$\text{Approval}_{\text{even}}$, $\text{Approval}_{\text{odd}}$}	{2-Approval, 1-Veto} [Thm. 6.3]
easy + hard = easy	{NoWinner, Approval}	{Majority, Bucklin} [Thm. 6.4]
easy + hard = hard	{AllWinner, Approval}	{1-Approval, 3-Veto} [Thm. 6.6]
hard + hard = easy	{ResoluteApproval, ResoluteApproval′}	{Condorcet_1, Condorcet_2} [Thm. 6.5]
hard + hard = hard	{Approval, Approval}	{4-Approval, 3-Veto} [Thm. 6.6]

Table 1: Results instantiating each easy-or-hard + easy-or-hard = easy-or-hard case. Easy refers to polynomial time and hard refers to NP-completeness. For each control example, we give the theorem establishing it.

THEOREM 5.3. *Let $\mathscr{F}$ be a finite set of voting rules such that every $f \in \mathscr{F}$ satisfies the following three conditions.*

1. *f is monotone,*
2. *f has the property that the score of a candidate $c \in C$ is polynomial-time computable if we know which candidates are ranked above and below c in each vote, and*
3. *the winners under f are the candidates with the highest score.*

Then $\mathscr{F}$-SB is solvable in polynomial time.

Theorem 5.3 follows immediately using Proposition 5.2, the downward closure of P under $\leq^p_{dtt}$ reductions, and the first theorem—changed to the nonunique-winner model, for which it does still hold—of Section 4 of [8] (a theorem that itself is drawing on a famous greedy-algorithm result of [2]).

6. CONTROL

In this section, we establish the results from the "Constructive Control by Adding Voters" column of Table 1. In addition, we classify the complexity of CCAV under voting-rule uncertainty for almost all finite families of k-Approval and k-Veto rules (see Classification 6.9).

By simple greedy algorithms (see, for example, [22]), constructive control by adding voters can be seen to be in P for the voting rules 1-Approval, 2-Approval, and 1-Veto. Unfortunately, we neither know of nor could we construct a greedy algorithm to handle the family {1-Approval, 2-Approval} or the family {1-Approval, 1-Veto}. Nonetheless, as Theorems 6.1 and 6.2 we resolve these cases, using the powerful modeling machinery provided by flow networks.

THEOREM 6.1. {1-Approval, 2-Approval}-CCAV *is in P.*

Proof. Consider an election $E = (C, V \cup W)$ with $C = \{p, c_1, \ldots, c_m\}$, distinguished candidate p, the collection V of registered voters, the collection W of unregistered voters, and a nonnegative integer k as the addition limit. Furthermore, let $W_1 = \{w \in W \mid p \text{ is first in } w\}$ and let $W_2 = \{w \in W \mid p \text{ is second in } w\}$. W.l.o.g., we will add only voters from $W_1 \cup W_2$. We also assume that we have at least two candidates.

We will loop over all $k_1, k_2 \geq 0$ such that $k_1 + k_2 \leq k$ and will see whether p can be made a winner by adding k_1 voters from W_1 and k_2 voters from W_2. Note that k_1 and k_2 fix the score of p both under 1-Approval and 2-Approval. For each pair of k_1 and k_2 we construct a flow network as displayed in Figure 1. We will define the remaining edge capacities below so that an (integral) flow of value $k_1 + k_2$ from s to t corresponds to a successful addition of voters and vice versa.

Our intention is that the value of the flow from p_1 to $c_{i,2}$ corresponds to the number of added voters with preference $p > c_i > \cdots$

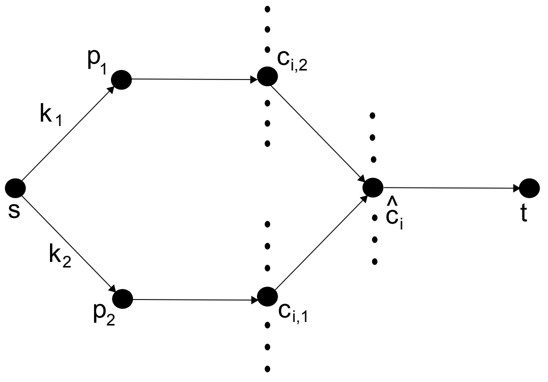

Figure 1: Network in the proof of Theorem 6.1

and that the value of the flow from p_2 to $c_{i,1}$ corresponds to the number of added voters with preference $c_i > p > \cdots$. If that is the case, the value of the flow from $\hat{c}_i$ to t corresponds to the number of added voters that 2-approve of c_i. In order to ensure that a flow of value $k_1 + k_2$ corresponds to a successful addition of voters, we set the capacities as follows.

For each i, $1 \leq i \leq m$, the edge from p_1 to $c_{i,2}$ has as its capacity the number of voters in W voting $p > c_i > \cdots$ and the edge from p_2 to $c_{i,1}$ has as its capacity the number of voters in W voting $c_i > p > \cdots$. This ensures that a flow of value $k_1 + k_2$ corresponds to an addition of k_1 voters from W_1 and k_2 voters from W_2. We still must ensure that this addition makes p a winner, i.e., for every i, $1 \leq i \leq m$, c_i does not have more 1-approvals than p and c_i does not have more 2-approvals than p. Note that the number of 1-approvals of p is $score^1_V(p) + k_1$ and that the number of 2-approvals of p is $score^2_V(p) + k_1 + k_2$.

To handle the 1-approvals, we let the capacity of the edge from $c_{i,1}$ to $\hat{c}_i$ be $score^1_V(p) + k_1 - score^1_V(c_i)$. To handle the 2-approvals, we let the capacity of the edge from $\hat{c}_i$ to t be $score^2_V(p) + k_1 + k_2 - score^2_V(c_i)$. (If any capacity is negative, there is no solution. We allow capacities to be 0.) The remaining edges do not have capacity restrictions. If we want to be explicit, we can set their capacities to $k_1 + k_2$.

From the discussion above, it is immediate that if there exists a flow of value $k_1 + k_2$, then we can make p a winner under 1-Approval and 2-Approval. It is also easy to see that making p a winner by adding k_1 voters from W_1 and k_2 voters from W_2 corresponds to a flow of value $k_1 + k_2$ in the network described above.

Since computing the maximum flow in a network is in polynomial time and the number of networks and their sizes are also bounded by a polynomial, we have solved our problem in polynomial time. ❑

THEOREM 6.2. {1-Approval, 1-Veto}-CCAV *is in P.*

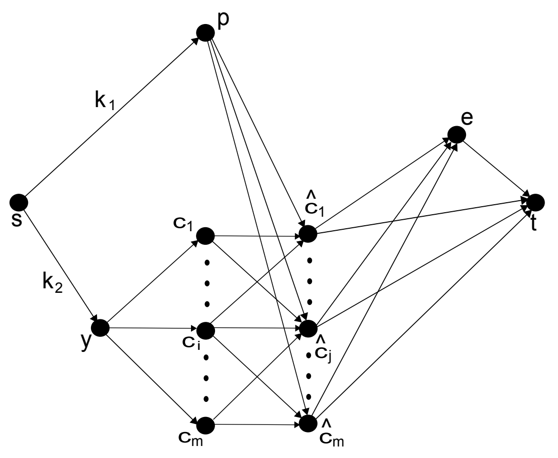

Figure 2: Network in the proof of Theorem 6.2

Proof. Consider an election $E = (C, V \cup W)$ with $C = \{p, c_1, \ldots, c_m\}$, distinguished candidate p, the collection V of registered voters, the collection W of unregistered voters, and a nonnegative integer k as the addition limit. W.l.o.g., we assume that we will never add voters that veto p. We also assume that we have at least two candidates. For every $k_1, k_2 \geq 0$ such that $k_1 + k_2 \leq k$, we will check whether we can make p a winner by adding k_1 voters that rank p first and k_2 voters that rank p neither first nor last. Note that k_1 and k_2 fix the 1-Approval and 1-Veto scores of p. Similarly to the proof of Theorem 6.1, for each k_1 and k_2, we construct a flow network as displayed in Figure 2. We will define the edge capacities below such that an (integral) flow of value $k_1 + k_2$ from s to t in the network corresponds to a successful addition of voters.

Our intention is that the value of the flow from p to $\hat{c}_j$ corresponds to the number of added voters that rank p first and c_j last and that the value of the flow from c_i to $\hat{c}_j$ corresponds to the number of added voters that rank c_i first and c_j last. In order to ensure that a flow of value $k_1 + k_2$ corresponds to a successful addition, we set the capacities as follows.

For each j, $1 \leq j \leq m$, the edge from p to $\hat{c}_j$ has as its capacity the number of voters in W that rank p first and c_j last. For each i, j, $1 \leq i, j \leq m$, the edge from c_i to $\hat{c}_j$ has as its capacity the number of voters in W that rank c_i first and c_j last. This ensures that a flow of value $k_1 + k_2$ corresponds to an addition of k_1 voters that rank p first and k_2 voters that rank p neither first nor last. We still must ensure that this addition makes p a winner, i.e., for every i, $1 \leq i \leq m$, c_i does not have more 1-approvals than p and for every j, $1 \leq j \leq m$, c_j has at least as many 1-vetoes as p. Note that the number of 1-approvals of p is $score_V^1(p) + k_1$ and the number of 1-vetoes of p is $vetoes_V^1(p)$.

To handle the approvals, we simply let the capacity of the edge from y to c_i be $score_V^1(p) + k_1 - score_V^1(c_i)$. (If there are negative capacities, there is no solution.) To handle the vetoes, we let the capacity of the edge from $\hat{c}_j$ to t be the number of 1-vetoes that c_j needs to get from the added voters, i.e., $\max(0, vetoes_V^1(p) - vetoes_V^1(c_j))$. In order to make p a 1-Veto winner, all these edges from $\hat{c}_j$ to t must be saturated. Of course, it is possible that c_j receives more vetoes. We capture these excess vetoes with vertex e. If the flow in the network has value $k_1 + k_2$, then the number of excess vetoes is exactly $k_1 + k_2 - \sum_{1 \leq j \leq m} \max(0, vetoes_V^1(p) - vetoes_V^1(c_j))$. We set the capacity of the edge from e to t to that value. This implies that if we have a flow of value $k_1 + k_2$, then all edges from $\hat{c}_j$ to t are saturated.

From the discussion above, it is immediate that if there exists a flow of value $k_1 + k_2$, then we can make p a winner under 1-Approval and 1-Veto. It is also easy to see that making p a winner by adding k_1 voters that rank p first and k_2 voters that rank p neither first nor last corresponds to a flow of value $k_1 + k_2$ in the network described above.

Since computing the maximum flow in a network is in polynomial time and the number of networks and their sizes are also bounded by a polynomial, we have solved our problem in polynomial time. ❑

Theorem 6.3 provides the case used in Table 1 to show an example where combining two easy-to-control rules yields a computationally hard case. And as part of our broader classification of cases, Theorem 6.7 provides other such cases.

THEOREM 6.3. $\{$2-Approval, 1-Veto$\}$-CCAV *is NP-complete.*

Proof. We now define the NP-complete problem 3-DIMENSIONAL MATCHING [16].

3-DIMENSIONAL MATCHING (3DM)	
Given:	Set $M \subseteq W \times X \times Y$, where W, X, and Y are disjoint sets having the same number k of elements.
Question:	Does M contain a matching, i.e., a subset $\widehat{M} \subseteq M$ such that $\|\widehat{M}\| = k$ and no two elements in $\widehat{M}$ agree in any coordinate?

We will construct a (many-one polynomial-time) reduction from 3DM to $\{$2-Approval, 1-Veto$\}$-CCAV. Let M be a 3DM instance. $M \subseteq W \times X \times Y$, $W = \{w_1, \ldots, w_k\}$, $X = \{x_1, \ldots, x_k\}$, and $Y = \{y_1, \ldots, y_k\}$. Construct the following CCAV instance. The candidate set is $C = W \cup X \cup Y \cup \{p\}$. p is the distinguished candidate. The addition limit is k. The unregistered voter collection consists of the following votes: For each $(w, x, y) \in M$, there is one vote of the form $w > x > p > \cdots > y$. The registered voter collection V consists of the following $2k + 1$ votes:

1. For each i, $1 \leq i \leq k-1$, there is one vote of the form $w_i > y_i > p > \cdots > x_i$.
2. For each i, $1 \leq i \leq k-1$, there is one vote of the form $x_i > y_i > p > \cdots > w_i$.
3. There is one vote of the form $p > y_k > \cdots > x_k$.
4. There is one vote of the form $p > y_k > \cdots > w_k$.
5. There is one vote of the form $x_k > w_k > \cdots > p$.

Note that for every $y \in Y$, y is not vetoed by the registered voters and the distinguished candidate p is vetoed exactly once. Furthermore, in (C, V) the 2-Approval score of each candidate in X and W is one, and the score of p and each candidate in Y is two. It is easy to show that there exists a 3-dimensional matching if and only if p can be made a winner of the election under both voting rules by adding at most k voters. ❑

Constructive control by adding voters is NP-complete for Bucklin elections [10]. Nonetheless, we have the following result, which holds not only for Bucklin but also for any voting rule f for which every Majority winner is also an f winner.

THEOREM 6.4. $\{$Majority, Bucklin$\}$-CCAV *is in P.*

Proof. It is immediately clear that constructive control by adding voters is in P for Majority. Since every Majority winner is also a Bucklin winner, the problem simplifies to constructive control by adding voters in Majority. ❑

THEOREM 6.5. *There are voting rules f_1 and f_2 such that f_1-*CCAV *and f_2-*CCAV *are NP-complete, but $\{f_1, f_2\}$-*CCAV *is in P.*

Proof. Let $f_1 =$ Condorcet$_1$ be the voting rule that if $\|C\| \equiv 0 \pmod 3$ simulates Plurality; if $\|C\| \equiv 1 \pmod 3$ simulates Condorcet; and if $\|C\| \equiv 2 \pmod 3$ all candidates lose. Let $f_2 =$ Condorcet$_2$ be the voting rule that if $\|C\| \equiv 0 \pmod 3$ simulates Plurality; if $\|C\| \equiv 1 \pmod 3$ all candidates lose; and if $\|C\| \equiv 2 \pmod 3$ it masks the lexicographically smallest candidate out of each of the votes and then simulates Condorcet among the remaining candidates.

It is not hard to see that Condorcet$_1$-CCAV is NP-complete (for example, by noting that in the NP-hardness proof of [3] for Condorcet-CCAV the image of the reduction always satisfies $\|C\| \equiv 1 \pmod 3$; that paper is in the unique-winner model rather than the nonunique-winner model, but for Condorcet, this makes no difference). It is also not hard to see that Condorcet$_2$-CCAV is NP-complete. However the election system in which each candidate c wins exactly if c is both a Condorcet$_1$ winner and a Condorcet$_2$ winner has the property that c wins exactly if $\|C\| \equiv 0 \pmod 3$ and c is a Plurality winner. And constructive control by adding voters is easily seen to be in P for this system, e.g., as a consequence of Plurality-CCAV belonging to P (in both the nonunique- and the unique-winner models in fact, though we need just the former). ❑

As mentioned previously, we are particularly interested in families of k-Approval and k-Veto elections (see Classification 6.9 for a summary of our results). Constructive control by adding voters for k-Approval is NP-complete for all $k \geq 4$ and is in P for $k < 4$, and constructive control by adding voters for k-Veto is NP-hard for all $k \geq 3$ and is in P for $k < 3$ [22]. We generalize this result as follows.

THEOREM 6.6. *For any sets $\{k_1, \ldots, k_\ell\} \subseteq \mathbb{N}$, with $k_1 < k_2 < \cdots < k_\ell$, and $\{\hat{k}_1, \ldots, \hat{k}_{\hat{\ell}}\} \subseteq \mathbb{N}$, with $\hat{k}_1 < \hat{k}_2 < \cdots < \hat{k}_{\hat{\ell}}$ the problem $\{k_1$-Approval$, \ldots, k_\ell$-Approval$, \hat{k}_1$-Veto$, \ldots, \hat{k}_{\hat{\ell}}$-Veto$\}$-CCAV is NP-complete if $k_\ell \geq 4$ or $\hat{k}_{\hat{\ell}} \geq 3$.*

This theorem gives plenty of examples of natural systems for which "easy + hard = hard" (e.g., $\{1$-Approval$, 3$-Veto$\}$ from Table 1) and "hard + hard = hard" (e.g., $\{4$-Approval$, 3$-Veto$\}$ from Table 1).

The only families of the general form $\{k_1$-Approval$, \ldots, k_\ell$-Approval$, \hat{k}_1$-Veto$, \ldots, \hat{k}_{\hat{\ell}}$-Veto$\}$ (with $\ell = 0$ and $\hat{\ell} = 0$ being acceptable possibilities) that are not already proven NP-hard by Theorem 6.6 are families that consist of some or all of these systems: $\{1$-Approval$, 2$-Approval$, 3$-Approval$, 1$-Veto$, 2$-Veto$\}$. Of course, all such families with exactly one system from that list are already known to be in P, and earlier theorems in this section have already proven certain of the two-rule subsets of this list to be in P or to be NP-complete. Let us continue our exploration of families built from this list.

The proof of Theorem 6.7 is similar to the proof of Theorem 6.3.

THEOREM 6.7. *Let $\mathscr{F} \subseteq \{1$-Approval$, 2$-Approval$, 3$-Approval$, 1$-Veto$, 2$-Veto$\}$.*

1. *If $\{2$-Approval$, 3$-Approval$\} \cap \mathscr{F} \neq \emptyset$ and $\{1$-Veto$, 2$-Veto$\} \cap \mathscr{F} \neq \emptyset$, the problem $\mathscr{F}$-*CCAV *is NP-complete.*

2. *If $\{1$-Approval$, 2$-Veto$\} \subseteq \mathscr{F}$, the problem $\mathscr{F}$-*CCAV *is NP-complete.*

THEOREM 6.8. $\{1$-Veto$, 2$-Veto$\}$-CCAV *is in P.*

Proof Sketch. Let us first explain why we did not handle this case earlier, when we were talking about network-flow algorithms at the start of this section. The reason is that even 2-Veto-CCAV is already too hard to handle with a basic network-flow algorithm. And the reason for this is that, as observed in [22], 2-Veto-CCAV is basically b-edge cover for multigraphs (and thus also basically b-edge matching for multigraphs) and these problems have complicated polynomial-time algorithms [15].

b-EDGE COVER FOR MULTIGRAPHS	
Given:	A multigraph $G = (V, E)$, a function b that assigns a nonnegative integer to every vertex, denoting the number of times that vertex needs to be covered, and a nonnegative integer k.
Question:	Does G have a b-edge cover of size k, i.e., a size-k submultiset E' of E such that every vertex v is incident to at least $b(v)$ edges in E'?

It is easy to see how to turn a 2-Veto-CCAV instance into an instance of b-edge cover for multigraphs. W.l.o.g. assume we add only voters that do not 2-veto p. That determines how many 2-vetoes we need to add for every candidate $c \neq p$. Define G as follows. Every candidate $c \neq p$ is a vertex, and $b(c)$ is the number of 2-vetoes for c that need to be added. For every voter whose last two candidates are $\{c, d\}$, we have an edge from c to d. It is immediate that p can be made a winner by adding k voters if and only if G has a b-edge cover of size k.

But $\{1$-Veto$, 2$-Veto$\}$-CCAV is a more difficult problem than 2-Veto-CCAV (much like $\{1$-Approval$, 2$-Approval$\}$-CCAV was more difficult than 2-Approval-CCAV) and does not correspond as nicely and directly to any edge-cover problem that we know of. Since the b-edge-cover algorithm is complicated, we have elected to not modify this algorithm directly, but instead to use it as a subroutine.

So, let C be the set of candidates, V the collection of registered voters, W the collection of unregistered voters, and p the distinguished candidate in our $\{1$-Veto$, 2$-Veto$\}$-CCAV instance. W.l.o.g., we assume that none of the voters in W ranks p last. The first difference with 2-Veto-CCAV is that we may need to add voters of the form $\cdots > p > c$. We will handle these voters first. Note that adding a voter of the form $\cdots > p > c$ is not very desirable. It is always better to add a voter of the form $\cdots > d > c$ with $d \neq p$, if one exists. Since we never add voters that rank p last, we know for every $c \neq p$ how many voters $v_1(c) \geq 0$ we need to add that rank c last. Now compute the number ℓ of voters in W that rank c last and that do not rank p second to last. If $\ell < v_1(c)$, we need to add $v_1(c) - \ell$ voters voting $\cdots > p > c$. We do this for every $c \neq p$. We add these voters to V and decrease the addition limit, k, accordingly. Now we have added all voters that rank p second to last that we need, and we delete all remaining such voters from W.

W.l.o.g., assume that $k \leq \|W\|$. None of the voters in W has p ranked in one of the last two positions, so we may without loss of generality assume that we add exactly k voters. For every $c \neq p$, let $v_1(c) \geq 0$ be the number of 1-vetoes for c we need to add, i.e., $v_1(c) = \max(0, vetoes^1_V(p) - vetoes^1_V(c))$ and let $v_2(c) \geq 0$ be the number of 2-vetoes for c we need to add.[1] We define $v_2(c)$ as $\max(v_1(c), vetoes^2_V(p) - vetoes^2_V(c))$, so that $v_2(c) \geq v_1(c)$. Note that p is a 1-Veto and 2-Veto winner in $(C, V \cup \widehat{W})$ if and only if for all $c \in C - \{p\}$, $vetoes^1_{\widehat{W}}(c) \geq v_1(c)$ and $vetoes^2_{\widehat{W}}(c) \geq v_2(c)$.

[1]We can not combine these two numbers into one and turn this basically into a weighted version of 2-Veto-CCAV, since that is NP-complete, even if we use only weights 1 and 2 [13].

To encode our problem into a graph, we start in the following way. For every candidate $c \neq p$, we have vertices c_1 and c_2. For every voter in W voting $\cdots > d > c$, we have an edge between c_1 and d_2. This encodes W. Our intention is that added voters correspond to edges in the edge cover. If 1-vetoes and 2-vetoes were independent of each other, that would be easy. But if for example $v_1(c) = 3$ and $v_2(c) = 5$, the edge cover could contain 5 edges incident to c_1 (i.e., 5 voters ranking c last), or 4 edges incident to c_1 and 1 edge incident to c_2 (i.e., 4 voters ranking c last and 1 voter ranking c second to last), or 3 edges incident to c_1 and 2 edges incident to c_2. We set $b(c_1) = 5$ and $b(c_2) = 2$. Note that the three edge covers described above are exactly the edge covers that contain an edge cover of size 5 that has the property that c_1 and c_2 together still need to be covered two times. We add an extra vertex $\hat{c}$ with $b(\hat{c}) = 2$, two edges between $\hat{c}$ and c_1, and two edges between $\hat{c}$ and c_2. It is easy to see (if we restrict our attention to candidate c) that a successful addition of 5 voters corresponds to a b-edge cover of size 5 + 2: if we add 5 voters that rank c last, we add 2 edges between $\hat{c}$ and c_2; if we add 4 voters ranking c last and 1 voter ranking c second to last, we add 1 edge between $\hat{c}$ and c_1 and 1 edge between $\hat{c}$ and c_2; and if we add 3 voters ranking c last and 2 voters ranking c second to last, we add 2 edges between $\hat{c}$ and c_1. If there are at least 5 voters in W that 2-veto c (and if there are fewer, control is not possible), a b-edge cover of size 5 + 2 can be transformed into a b-edge cover of size 5 + 2 that covers $\hat{c}$ *exactly* twice (simply replace extra edges between $\hat{c}$ and c_i by edges incident to c_i and not to $\hat{c}$), which in turn corresponds to a successful addition of 5 voters.

The complete construction of G works as follows.

1. First check that p is a winner if we add all of W (recall that W does not contain any voters that 2-veto p). If this is not the case, control is impossible, and we reject.

2. Define G as described above. Let $b(c_1) = v_2(c)$, $b(c_2) = v_2(c) - v_1(c)$, and $b(\hat{c}) = v_2(c) - v_1(c)$.

It can be shown that a successful control action is possible if and only if G has a b-edge cover of size $k + \sum_{c \neq p} b(\hat{c})$. ❑

The following classification summarizes our results on the complexity of CCAV under voting-rule uncertainty for families of k-Approval and k-Veto elections.

CLASSIFICATION 6.9. *Let $\mathscr{F}$ be a finite family of k-*Approval *and k-*Veto *elections of size at least two. We are in one of the following cases.*

1. *$\mathscr{F}$-*CCAV *is NP-complete by Theorems 6.6 or 6.7.*

2. $\mathscr{F} = \{$1-Approval, 2-Approval$\}$. *Then $\mathscr{F}$-*CCAV *is in P by Theorem 6.1.*

3. $\mathscr{F} = \{$1-Approval, 1-Veto$\}$. *Then $\mathscr{F}$-*CCAV *is in P by Theorem 6.2.*

4. $\mathscr{F} = \{$1-Veto, 2-Veto$\}$. *Then $\mathscr{F}$-*CCAV *is in P by Theorem 6.8.*

5. $\mathscr{F} = \{$1-Approval, 3-Approval$\}$ *or* $\mathscr{F} = \{$2-Approval, 3-Approval$\}$ *or* $\mathscr{F} = \{$1-Approval, 2-Approval, 3-Approval$\}$. *These cases are still open.*

One may wonder why the approach from Theorem 6.8 doesn't seem to work for, for example, $\{$1-Approval, 3-Approval$\}$-CCAV. The crucial difference is that in that case there are two types of voters to consider, namely voters of the form $p > \{c, d\} > \cdots$ and voters of the form $c > \{p, d\} > \cdots$. And adding these different types of voters has a different effect on the scores of p (and thus also on the number of 1-approvals and 3-approvals that the other candidates can get). This is in contrast to the $\{$1-Veto, 2-Veto$\}$-CCAV proof, where after some preprocessing, we had to consider voters of only one type, namely voters of the form $\cdots > c > d$, with $p \notin \{c, d\}$.

Finally, we briefly mention that control is sometimes analyzed not just in hard/easy terms, but in a 3-part analysis whose options are immune, vulnerable, and resistant [3]. Although we omit the definitions and details, we mention in passing that we have completely analyzed, for CCAV which of these three possibilities can be realized from each way of combining one of these three with one of these three. (For example, when $\mathscr{F}$ consists of one vulnerable rule and one resistant rule, we have examples realizing each of immune, vulnerable, and resistant.) However, we consider this far less interesting than the main analysis of this section (and the corresponding right column of Table 1 and Classification 6.9) and its stress is on natural systems.

7. DESTRUCTIVE ACTIONS

So far we investigated only constructive versions of our problems, i.e., our goal was to make a candidate win the election. We now explain why that is so: The "destructive" case is not too interesting to study, as it decomposes into its underlying single-rule questions.

One natural notion of destruction would be to preclude constructive success. So since constructive success for a family of rules means winning in all of them, destructive success would mean ensuring that the designated candidate is not a winner under at least one rule in the family. That is, we have the following definition. (For rest of this section, when speaking of actions $\mathscr{A}$ we do not include constructive/destructive as part of that action, and so which of constructive or destructive holds must be specified.)

DEFINITION 7.1. *Let $\mathscr{A}$ be any standard type of bribery, control, or manipulation. Let $\mathscr{F}$ be a family of voting rules. $\mathscr{F}$-*DESTRUCTIVE-*$\mathscr{A}$ is defined as the set of all instances x such that* $(\exists f_i \in \mathscr{F})[x \in f_i$-DESTRUCTIVE-$\mathscr{A}]$.

This definition is also precisely what one would get as one's destructive notion if one thinks of the constructive cases as being defined not in the way we defined them earlier in this paper, but instead in the following equivalent way. We can view a constructive problem $\mathscr{F}$-CONSTRUCTIVE-$\mathscr{A}$ as being the problem f-CONSTRUCTIVE-$\mathscr{A}$, where f is the election rule whose winners on a given instance are precisely the candidates who on that instance are winners under every election rule in $\mathscr{F}$.

Under Definition 7.1, each destructive action on a family of voting rules simply decomposes into the underlying single-rule destructive-action cases. Basically, for a successful destructive action we only have to be (destructively) successful under one voting rule. It follows easily from this that P results for the underlying systems are inherited for the uncertain election case, as the following results makes explicit.

THEOREM 7.2. *Let $\mathscr{A}$ be any standard type of bribery, control, or manipulation. Let $\mathscr{F}$ be a finite family of voting rules. $\mathscr{F}$-*DESTRUCTIVE-*$\mathscr{A} \in P$ if* $(\forall f_i \in \mathscr{F})[\, f_i$-DESTRUCTIVE-$\mathscr{A} \in P]$.

The sharp-eyed reader will note two surprises here. Unlike the definition, this is an "if" rather than an "if and only if." And we have limited this claim to *finite* families. Both of these limitations are needed. It is not hard to construct counterexamples to the "only if" version, and to the "infinite families" version, of the above result.[2]

Finally, as an open direction (in addition to the still-open Case 5 of Classification 6.9 and the direction of allowing the rule-chooser independent preferences so as to make this an interesting game-theoretic situation), we mention that another very interesting notion of destruction to study would be to seek to ensure that a given candidate fails to win under all the rules in $\mathscr{F}$. This alternate notion is what one would frame if one's goal were to utterly destroy a candidate's chances no matter what system from $\mathscr{F}$ is employed, and so is the right notion to use if the manipulative-action doer moves first and hates the designated candidate and the election-system chooser moves next and loves the designated candidate. And Definition 7.1 is what one should employ if one is studying the complexity of destroying the goal of the constructive case (which is having the candidate win in all systems), and is the right notion to use if the manipulative-action doer moves first and hates the designated candidate and the election system chooser moves next and also hates the designated candidate. Which is the right definition to study will depend on the particular real-world dynamics that one is trying to model.

Acknowledgments We are grateful to the AAMAS referees for helpful suggestions.

8. REFERENCES

[1] Y. Bachrach, N. Betzler, and P. Faliszewski. Probabilistic possible winner determination. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, pages 697–702. AAAI Press, July 2010.

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

[3] J. Bartholdi, III, C. Tovey, and M. Trick. How hard is it to control an election? *Mathematical and Computer Modeling*, 16(8/9):27–40, 1992.

[4] D. Baumeister, M. Roos, and J. Rothe. Computational complexity of two variants of the possible winner problem. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems*, pages 853–860, May 2011.

[5] N. Betzler and B. Dorn. Towards a dichotomy of finding possible winners in elections based on scoring rules. *Journal of Computer and System Sciences*, 76(8):812–836, 2010.

[6] V. Conitzer, T. Walsh, and L. Xia. Dominating manipulations in voting with partial information. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, pages 638–643. AAAI Press, Aug. 2011.

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

[8] E. Elkind and G. Erdélyi. Manipulation under voting rule uncertainty. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, pages 624–634, June 2012.

[9] E. Ephrati and J. Rosenschein. A heuristic technique for multi-agent planning. *Annals of Mathematics and Artificial Intelligence*, 20(1–4):13–67, 1997.

[10] G. Erdélyi, L. Piras, and J. Rothe. The complexity of voter partition in Bucklin and fallback voting: Solving three open problems. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems*, pages 837–844, May 2011.

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

[12] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. How hard is bribery in elections? *Journal of Artificial Intelligence Research*, 35:485–532, 2009.

[13] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. Weighted electoral control. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems*, pages 367–374, May 2013.

[14] P. Faliszewski, E. Hemaspaandra, and H. Schnoor. Copeland voting: Ties matter. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems*, pages 983–990, May 2008.

[15] H. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proc. of the 18th ACM Symposium on Theory of Computing*, pages 448–456. ACM Press, May 1986.

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

[17] S. Ghosh, M. Mundhe, K. Hernandez, and S. Sen. Voting for movies: The anatomy of recommender systems. In *Proceedings of the 3rd Annual Conference on Autonomous Agents*, pages 434–435. ACM Press, May 1999.

[18] E. Hemaspaandra, L. Hemaspaandra, and C. Menton. Search versus decision for election manipulation problems. Technical Report arXiv:1202.6641 [cs.GT], arXiv.org/corr/, Feb. 2012. Revised, March 2012.

[19] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Anyone but him: The complexity of precluding an alternative. *Artificial Intelligence*, 171(5–6):255–285, 2007.

[20] K. Konczak and J. Lang. Voting procedures with incomplete preferences. In *Proceedings of the Multidisciplinary IJCAI-05 Workshop on Advances in Preference Handling*, pages 124–129, July/August 2005.

[21] R. Ladner, N. Lynch, and A. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1(2):103–124, 1975.

[22] A. Lin. *Solving Hard Problems in Election Systems*. PhD thesis, Rochester Institute of Technology, 2012.

[23] S. Obraztsova, E. Elkind, and N. Hazon. Ties matter: Complexity of voting manipulation revisited. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems*, pages 71–78, May 2011.

[2]For example, to swat down the "only if" version, we can have two voting rules each covering half of the domain (e.g., for destructive control by adding voters we can have a voting rule f_1, under which each candidate loses if $\|C\|$ is even and otherwise f_1-DCAV is NP-complete, and a voting rule f_2 under which each candidate loses if $\|C\|$ is odd and otherwise f_2-DCAV is NP-complete, and this already breaks the "only if" version). As to the case of infinite families, one can also build a counterexample. We won't do so here due to space, but the example involves embedding (non)winner testing into manipulative actions, and using the facts that there are only a countably infinite number of sets in P, yet even infinite families of very simple rules can create uncountably many settings that need to be correctly handled.