The Complexity of Bribery in Elections[1]

Piotr Faliszewski, Edith Hemaspaandra, and
Lane A. Hemaspaandra

Abstract

We study the complexity of influencing elections through bribery: How computationally complex is it for an external actor to determine whether by a certain amount of bribing voters a specified candidate can be made the election's winner? We study this problem for election systems as varied as scoring protocols and Dodgson voting, and in a variety of settings regarding the nature of the voters, the size of the candidate set, and the specification of the input. We obtain both polynomial-time bribery algorithms and proofs of the intractability of bribery. Our results indicate that the complexity of bribery is extremely sensitive to the setting. For example, we find settings where bribing weighted voters is NP-complete in general but if weights are represented in unary then the bribery problem is in P. We provide a complete classification of the complexity of bribery for the broad class of elections (including plurality, Borda, k-approval, and veto) known as scoring protocols.

1 Introduction

This paper studies the complexity of bribery in elections, that is, the complexity of computing whether it is possible, by modifying the preferences of a given number of voters, to make some preferred candidate a winner. Recall that an election system provides a framework for aggregating voters' preferences—ideally (though there is no truly ideal voting system [DS00, Gib73, Sat75]) in a way that is satisfying, attractive, and natural. Societies use elections to select their leaders, establish their laws, and decide their policies. However, practical applications of elections are not restricted to people and politics. Many parallel algorithms start by electing leaders; multiagent systems sometimes use voting for the purpose of planning [ER93]; web search engines can aggregate results using methods based on elections [DKNS01].

With such a range of applications, it is not surprising that elections may have a wide range of voter-to-candidate proportions. For example, in typical presidential elections there are relatively few candidates but there may be millions of voters. In the context of the web, one may consider web pages as voting on other pages by linking to them, or may consider humans to be voting on pages at a site by the time they spend on each. In such a setting we may have both a large number of voters and a large number of candidates. On the other

[1]A preliminary version of this paper appeared in the Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06); a full version is also available [FHH06]. Supported in part by NSF grants CCR-0311021 and CCF-0426761, a Friedrich Wilhelm Bessel Research Award, and the Alexander von Humboldt Foundation's TransCoop program.

hand, Dwork et al. [DKNS01] suggest designing a meta search engine that treats other search engines as voters and web pages as candidates. This yields very few voters but many candidates.

Typically, we are used to the idea that each vote is equally important. However, all the above scenarios make just as much sense in a setting in which each voter has a different voting power. For example, U.S. presidential elections are in some sense weighted (different states have different voting powers in the Electoral College); shareholders in a company have votes weighted by the number of shares they own; and search engines in the above example could be weighted by their quality. Weighted voting is a natural choice in many other settings as well.

The importance of election systems naturally inspired questions regarding their resistance to abuse, and several potential dangers were identified and studied. For example, the organizers can make attempts to *control* the outcome of the elections by procedural tricks such as adding or deleting candidates or encouraging/discouraging people from voting. Classical social choice theory is concerned with the possibility or impossibility of such procedural control. However, recently it was realized that even if control is possible, it may still be difficult to find what actions are needed to effect control, e.g., because the computational problem is NP-complete. The complexity of controlling who wins the election was first studied by Bartholdi, Tovey, and Trick [BTT92].

Elections are endangered not only by the organizers but also by the voters (*manipulation*), who might be tempted to vote strategically (that is, not according to their true preferences) to obtain their preferred outcome. This is not desirable as it can skew the result of the elections in a way that is arguably not in the best interest of the society. The Gibbard–Satterthwaite/Duggan–Schwartz Theorems [Gib73, Sat75, DS00] show that essentially all election systems can be manipulated. So it is important to discover for which systems manipulation is *computationally difficult* to execute. This line of research was started by Bartholdi, Tovey, and Trick [BTT89a], and was continued by many researchers, e.g, [CS02a, CS02b, CS03, CLS03, EL05, HH05].

Surprisingly, nobody seems to have addressed the issue of (the complexity of) bribery, i.e., attacks where the person interested in the success of a particular candidate picks a group of voters and convinces them to vote as he or she says. Bribery seems strongly motivated from both real life and from computational agent-based settings, and shares some of the flavor of both manipulation (changing voters' (reported) preferences) and control (deciding which voters to influence). This paper initiates the study of the complexity of bribery in elections.

There are many different settings in which bribery can be studied. In the simplest one we are interested only in the least number of voters we need to bribe to make our favored candidate win. A natural extension is to consider prices for each voter. In this setting, voters are willing to change their true preferences to anything we say, but only if we can meet their price. In an even more complicated setting it is conceivable that voters would have different prices depending on how we want to affect their vote (however, it is not clear how to succinctly encode a voter's price scheme). We study only the previous two scenarios.

We classify election systems with respect to bribery by in each case seeking to either prove the complexity is low by giving a polynomial-time algorithm or argue intractability via proving the NP-completeness of discovering whether bribery can affect a given case. We obtain a broad range of results showing that the complexity of bribery depends closely on the setting. For example, for weighted plurality elections, bribery is in P but jumps to being NP-complete if the weighted voters have price tags as well. As another example, for approval voting the manipulation problem is easily seen to be in P, but in contrast we prove that the bribery problem is NP-complete. Yet we also prove that when the bribery cost function is made more local the complexity of approval voting falls back to P. For scoring protocols we obtain a full classification of the complexity of bribery, for all the settings that we consider.

The paper is organized as follows. In the preliminary section we describe the election systems and bribery problems we are interested in. Then we provide a detailed study of bribery in plurality elections. After that we study connections between manipulation and bribery, and fully classify bribery under scoring protocols. Finally, we study the case of succinctly represented elections. Due to space limits, the proofs are omitted except for some brief sketches. All details can be found in the full version [FHH06].

2 Preliminaries

We can describe elections by providing a set $C = \{c_1, \ldots, c_m\}$ of candidates, a set V of voters specified by their preferences, and a rule for selecting winners. A voter v's preferences are represented as a list $c_{i_1} > c_{i_2} > \ldots > c_{i_m}$, $\{i_1, i_2, \ldots, i_m\} = \{1, 2, \ldots, m\}$, where c_{i_1} is the most preferred candidate and c_{i_m} is the most despised one. We assume that preferences are transitive, complete (for every two candidates each voter knows which one he or she prefers), and strict.

Let us briefly describe the election systems that we analyze in this paper.[2] Winners of *plurality* elections are the candidate(s) who are the top choice of the largest number of voters (of course, these will be different voters for different winners). In *approval* voting each voter selects candidates he or she approves of; the candidate(s) with the most approvals win.

A *scoring protocol* for m candidates is described by a vector $\alpha = (\alpha_1, \ldots, \alpha_m)$ of nonnegative integers such that $\alpha_1 \geq \alpha_2 \ldots \geq \alpha_m$. (We have not required

[2]In the social choice literature, often voting systems are assumed to have at least one winner, or exactly one winner, but at least in terms of the notion of voting system, we do not require such a restriction, since one can imagine wanting to study elections in which—perhaps due to tie effects or symmetry effects (or even due to having zero candidates)—there is not always exactly one winner. Indeed, in practice, in such elections as those on Hall of Fame induction worthiness or on who should be hired at a given academic department, it is quite possible that a real-world election system might give the answer "No one this year."

$\alpha_1 > \alpha_m$, as we wish to classify the broadest class of cases possible, including the usually easy boundary case when all α_i's are equal.) Each time a candidate appears in the i'th position of a voter's preference list, that candidate gets α_i points; the candidate(s) who receive the most points win. Well-known examples of scoring protocols include the Borda count, plurality, k-approval, and veto voting systems, where for m-candidate elections Borda count uses $\alpha = (m-1, m-2, \ldots, 0)$, plurality uses $\alpha = (1, 0, \ldots, 0, 0)$, k-approval uses $(1^k, 0^{m-k})$, and veto uses $\alpha = (1, 1, \ldots, 1, 0)$.

A Condorcet winner is a candidate who (strictly) beats all other candidates in pairwise contests, that is, a Condorcet winner beats everyone else in pairwise plurality elections. Clearly, there can be at most one Condorcet winner, but sometimes there are none. There are many voting systems that choose the Condorcet winner if one exists and use some compatible rule otherwise. One such system is that of Dodgson, where a winner is the person(s) who can become a Condorcet winner by a smallest number of switches in voters' preference lists. (A switch changes the order of two adjacent candidates on a list.) If a Condorcet winner exists, he or she is the unique winner in Dodgson's scheme. See Dodgson [Dod76] for details regarding Dodgson's voting rule, under which it is known that winner testing is complete for parallel access to NP [HHR97].

Now let us define the bribery problem for a given election system $\mathcal{E}$. All numbers are nonnegative integers and, unless otherwise specified, are represented in binary. $\mathcal{E}$-bribery is the following problem.

Given: A set C of candidates, a set V of voters specified via their preference lists, distinguished candidate p, and a nonnegative integer k.

Question: Is it possible to make p a winner of the $\mathcal{E}$ election by changing the preference lists of at most k voters?

Bribery problems come in several different flavors. In the unweighted case, the default for this paper, all voters are equal; in the weighted case each voter has a possibly different weight. In the $\mathcal{E}$-\$bribery family of problems we assume that each voter has a price for changing his or her preference list. In such a case we ask not whether we can bribe at most k people, but whether we can make p a winner by spending at most k dollars on bribing. Naturally, we also consider \$bribery problems with weighted voters.

Regarding the fact that in these models voters are assumed to vote as the bribes dictate, we stress that by using the term bribery, we do not intend to imply any moral failure on the part of bribe recipients: Bribes are simply payments.

Formally, our bribery problems speak of making the preferred candidate a winner rather than making him or her the unique winner. However, essentially all our results hold for the unique winner cases as well.

As always, we say $A \leq_m^p B$ (A many-one polynomial-time reduces to B) if there is a polynomial-time computable function f such that $x \in A \iff f(x) \in B$. We also use disjunctive truth-table reductions: $A \leq_{dtt}^p B$ (A disjunctively truth-table reduces to B) if there is a polynomial-time procedure that on input

x outputs a list of strings such that $x \in A$ if and only if at least one of those strings is in B. See, e.g., the work of Ladner, Lynch, and Selman [LLS75] for details regarding various reduction types. $\|S\|$ denotes the cardinality of set S.

3 Plurality

In this section we establish the complexity of bribery for plurality rule elections. The widespread use of plurality elections makes these results of particular relevance.

Not surprisingly, plurality-bribery is easy.

Theorem 3.1 plurality-bribery *is in P.*

To make sure our favorite candidate p wins, it is enough to keep on bribing voters of the currently most popular candidate to vote for p until p becomes a winner. However, bribery within the plurality system is not always easy.

Theorem 3.2 plurality-weighted-\$bribery *is NP-complete, even for just two candidates.*

That is, bribery is easy in the simplest case, but if we allow voters to have prices and weights, then the problem becomes intractable. It is natural to ask which of the additional features (prices? weights?) is responsible for making the problem difficult. It turns out that neither of them is the sole reason and that only their combination yields enough power to make the problem NP-complete.

Theorem 3.3 *Both* plurality-\$bribery *and* plurality-weighted-bribery *are in P.*

A direct greedy algorithm, like that underpinning Theorem 3.1, fails to prove Theorem 3.3. Rather we approach Theorem 3.3's proof as follows. Assume that p will be capable of getting at least r votes (or in the weighted case, r vote weight), where r is some number to be specified later. If this is to make p a winner, we need to make sure that everyone else gets at most r votes. Thus we carefully choose enough cheapest (heaviest) voters of candidates that defeat p and bribe those voters to vote for p. Then we simply have to make sure that p gets at least r votes by bribing the cheapest (the heaviest) of the remaining voters. If during this process p ever becomes a winner without exceeding the budget (the bribe limit) then we know that bribery is possible. How do we pick the value of r? In the case of plurality-\$bribery, we can just run this procedure for all $\|V\|$ possible values, and accept exactly if it succeeds for at least one of them. For plurality-weighted-bribery a slightly trickier approach works. (Essentially, we need to try only $\|V\|$ values as well; we can start from $r = 1$ and always increase r so that minimally less people need to be bribed in the first part of the above algorithm.)

Pushing the approach outlined above even further it is in fact possible to get yet stronger results. In plurality-weighted-\$bribery we assume that both prices and weights are encoded in binary. However, if either the prices or the weights are encoded in unary, then the problem becomes easy.

Theorem 3.4 *Both* plurality-weighted-\$bribery$_{\text{unary}}$ *and* plurality-weighted$_{\text{unary}}$-\$bribery *are in P.*

The main idea of the proof of Theorem 3.4 is the same as that underpinning the proof of Theorem 3.3, but the details are more complicated. Theorem 3.4 is particularly interesting because it says that plurality-weighted-\$bribery will be difficult only if we choose both weights and bribe prices to be high. But the prices are set by voters, and in many cases one could assume that there would be fairly low values for the bribe prices and so the problem would be easy.

We can look at the problem of bribery within plurality elections from a yet another perspective. Note that all of the above algorithms (in most cases, implicitly) assume that we bribe others to vote for p. This is a reasonable method of bribing if one wants p to become a winner, but it also has potential real-world downsides: The more people we bribe, the more likely it may be that the malicious attempts will be detected and will work against p. To minimize the chances of that happening we might instead bribe voters not to vote for p but for some other candidates. This way p does not get extra votes but might be able to take away enough voters from the most popular candidates to become a winner. We call this setting negative-bribery. Negative bribery draws a very sharp line between the complexity of bribing weighted and priced voters.

Theorem 3.5 plurality-weighted-negative-bribery *is NP-complete, but* plurality-negative-\$bribery *is in P.*

4 Bribery versus Manipulation

The previous section provides a detailed discussion of the complexity of bribery for plurality voting. Its results are obtained by hand-crafting algorithms and reductions. It would be nicer if one could find tools that would let one inherit complexity results from the vast election systems literature. In this section we study relations between bribery and manipulation, and show how to obtain results using the relations we find. In the next section we will discuss another fairly general tool to study certain types of bribery and manipulation problems.

Manipulation is in flavor somewhat similar to bribery, with the difference that in manipulation the set of voters who may change their preference lists is specified by the input. Bribery can be viewed as manipulation where the set of manipulators is not fixed in advance and finding who to manipulate is part of the challenge. This might suggest that bribery problems should not be easier than analogous manipulation ones. In fact, there are election systems for which bribery is NP-complete but manipulation is easy.

Theorem 4.1 approval-bribery *is NP-complete, but* approval-manipulation *and* approval-weighted-manipulation *are both in P.*

Algorithms for approval-manipulation and approval-weighted-manipulation are trivial: The manipulating group approves of just the favorite candidate. The NP-completeness result follows from a reduction from the NP-complete Exact-Cover-by-3Sets problem.

Somewhat surprisingly, it is also possible that manipulation is NP-complete, while bribery is in P. We have designed an artificial election system where this is the case.

Theorem 4.2 *There exists a voting system $\mathcal{E}$ for which manipulation is NP-complete, but bribery is in P.*

We briefly sketch a proof of this theorem. Let A be an NP-complete set and let $B \in \mathrm{P}$ be such that

1. $A = \{x \in \Sigma^* \mid (\exists y \in \Sigma^*)[\langle x, y\rangle \in B]\}$, and
2. $(\forall x, y \in \Sigma^*)[\langle x, y\rangle \in B \Rightarrow |x| = |y|]$.

Such sets can easily be constructed from any NP-complete set by padding. The idea of the proof is to embed a verifier for A within the election rule $\mathcal{E}$. We do this in a way that forces manipulation to solve arbitrary A instances, while allowing bribery to still be easy.

First, we observe that preference lists can be used to encode arbitrary binary strings. We will use the following encoding. For C a set of candidates, let $c_1, c_2, \ldots, c_m$ be those candidates in lexicographical order. We will view the preference list

$$c_{i_1} > c_{i_2} > c_{i_3} > \cdots > c_{i_m}$$

as an encoding of the binary string $b_1 b_2 \cdots b_{\lfloor m/2 \rfloor}$, where for each j, $1 \leq j \leq \lfloor m/2 \rfloor$, if $i_{2j-1} > i_{2j}$ then $b_j = 0$ and otherwise $b_j = 1$.

In our reduction, binary strings starting with 1 will encode instances, and binary strings starting with 0 will encode witnesses. Given this setup, we can describe our election system $\mathcal{E}$. Let (C, V) be an election. For each $c \in C$, c is a winner of the election if and only if $\|V\| = 3$ and

Rule 1: all preference lists encode strings starting with 1 or all preference lists encode strings starting with 0, or

Rule 2: exactly one preference list encodes a string that starts with 1, say $1x$, and at least one other preference list encodes a string $0y$ such that $\langle x, y\rangle \in B$.

Thus, either all candidates are winners or none of them are winners. Note that testing whether a candidate c is a winner of an $\mathcal{E}$ election can easily be done in polynomial time. $\mathcal{E}$-bribery is in P because we are in one of the following three cases: all candidates are winners already, or no candidate can become a winner because $\|V\| \neq 3$, or we can make each candidate a winner by bribing exactly one voter so that all voters' preference lists encode strings that start with the same symbol.

On the other hand, the ability to solve the manipulation problem for $\mathcal{E}$ implies the ability to solve A. We construct a reduction from A to $\mathcal{E}$-manipulation. Given a string $x \in \Sigma^*$, we first check whether $\langle x, 0^{|x|} \rangle \in B$. If so, then clearly $x \in A$ and we output some fixed member of $\mathcal{E}$-manipulation. Otherwise, we output a manipulation problem with candidates $\{1, 2, \ldots, 2(|x|+1)\}$ and three voters, v_0, v_1, and v_2, such that v_0's preference list encodes $1x$, v_1's preference list encodes $00^{|x|}$, and v_2 is the only manipulative voter. We claim that candidate 1 can be made a winner if and only if $x \in A$.

Since $\langle x, 0^{|x|} \rangle \notin B$, the only way in which v_2 can make 1 a winner is when v_2 encodes a string $0y$ such that $\langle x, y \rangle \in B$ in which case $x \in A$. For the converse, if $x \in A$, there exists a string $y \in \Sigma^{|x|}$ such that $\langle x, y \rangle \in B$. We can encode string $0y$ as a preference list over $\{1, 2, \ldots, 2(|x|+1)\}$, and let this be the preference list for v_2. This ensures that 1 is a winner of the election.

Since this reduction can be computed in polynomial time, and the $\mathcal{E}$-manipulation's membership in NP is clear, we have that $\mathcal{E}$-manipulation is NP-complete.

While the above election system is not natural, together with the results on approval voting it tells us that we cannot hope to get a result that says "manipulation always reduces to an analogous bribery problem," unless P = NP. Nonetheless, if instead of looking at all election systems and all versions of bribery and manipulation we somewhat restrict our focus, either to some subclass of election systems or to some subclass of bribery types, then we can still find interesting connections between the complexity of bribery and the complexity of manipulation.

First, let us observe that to check whether bribery can be successful on a given input we can simply try all possible manipulations by k voters, where k is the number of bribes we are willing to make. This way for a fixed k we can disjunctively truth-table reduce any bribery problem to the analogous manipulation problem. In the following meta-theorem, $\mathcal{B}$ means any of our bribery problems that does not involve prices, and $\mathcal{M}$ represents the analogous manipulation problem.

Theorem 4.3 *For each fixed k it holds that $\mathcal{B} \leq_{dtt}^{p} \mathcal{M}$, where the bribery problem $\mathcal{B}$ allows at most k bribes, and the manipulation problem $\mathcal{M}$ allows the manipulator set to contain any number of voters between* 0 *and k.*

While simple, this result is still powerful enough to inherit some results from previous papers. Bartholdi, Tovey, and Trick [BTT89a] discuss manipulations by single voters. Theorem 4.3 translates their results to the bribery case. In particular, this translation says that bribery for $k = 1$ is in P for plurality, Borda count and many other systems.

Instead of looking at bribery problems that restrict the number of voters we can affect, we can focus on a restricted set of election systems. Namely, we will concentrate on a very natural and broad family, the scoring protocols.

Hemaspaandra and Hemaspaandra [HH05] showed a dichotomy theorem that classifies weighted manipulation problems for all scoring protocols as either being NP-complete or in P (see also [PR06] and the unpublished 2005 combined version of [CS02a, CLS03]). By reducing manipulation to bribery for scoring protocols (but also by employing several other insights) we have obtained an analogous classification for the case of bribery.

Theorem 4.4 *For each scoring protocol $\alpha = (\alpha_1, \dots, \alpha_m)$ Table 1 shows the complexity of each of the five natural bribery problems for that scoring protocol.*

We will now briefly sketch how Table 1 was obtained. First, let us note that for each scoring protocol $\alpha = (\alpha_1, \dots, \alpha_m)$ such that $\alpha_1 = \cdots = \alpha_m$ any bribery problem is in P; in this case each preference list has the same effect on elections. For $\alpha_1 > \alpha_2 = \cdots = \alpha_m$ it is clear that each appropriate bribery problem is a special case of an analogous bribery problem for plurality. (Only the NP-completeness result needs some care, but it can be translated to the scoring protocol world as well.) Thus, the interesting part of the table is that where it is not the case that $\alpha_2 = \cdots = \alpha_m$.

Each time we consider some scoring protocol $\alpha = (\alpha_1, \dots, \alpha_m)$ we automatically limit ourselves to a setting with a fixed constant number of candidates, namely m. Thus, there are only $m!$ different preference orders each voter might have, and so it is possible to evaluate all possible ways of bribing unweighted voters. This gives us that both α-bribery and α-\$bribery are in P for any α. Using a similar in spirit, but somewhat more involved, dynamic-programming algorithm we can also show that α-weighted$_{\text{unary}}$-\$bribery is in P.

It remains to show that α-weighted-bribery and α-weighted-\$bribery are NP-complete when it is not the case that $\alpha_2 = \cdots = \alpha_m$. In the case of α-weighted-\$bribery this is fairly easy as we simply need to observe that manipulation is in fact just a special case of \$bribery (Theorem 4.5) and invoke the Hemaspaandra and Hemaspaandra dichotomy theorem.

Theorem 4.5 *Let $\mathcal{M}$ be some manipulation problem and let $\mathcal{B}$ be the analogous \$bribery problem (for the same election system). It holds that $\mathcal{M} \leq_m^p \mathcal{B}$.*

The reduction simply takes an instance of the manipulation problem and outputs a bribery problem that is identical to the manipulation one only that all the nonmanipulators have price 1, all manipulators have price 0 and (just for specificity) some fixed preferences, and the budget is set to 0.

It remains to show that for scoring protocols α such that it is not the case that $\alpha_2 = \cdots = \alpha_m$ the α-weighted-bribery is still NP-complete, even without the use of price tags. It would be nice to do so by reducing to our problem from the corresponding manipulation problems. This seems not to work, but we construct such a reduction that has the right properties whenever its inputs satisfy an additional condition (namely, that the weight of the lightest manipulating voter is at least double that of the heaviest nonmanipulator). So we would be done if this restriction of the manipulation problem were NP-hard. To show that, we by close examination of the dichotomy proof of Hemaspaandra and Hemaspaandra [HH05] prove that though the reduction from partition to that manipulation problem does not obey the desired condition in all its image

	Scoring protocol $\alpha = (\alpha_1, \ldots, \alpha_m)$.		
		$\alpha_1 > \alpha_2$ and	not true that
bribery problem	$\alpha_1 = \cdots = \alpha_m$	$\alpha_2 = \cdots = \alpha_m$	$\alpha_2 = \cdots = \alpha_m$
α-bribery	P	P	P
α-\$bribery	P	P	P
α-weighted$_{\text{unary}}$-\$bribery	P	P	P
α-weighted-bribery	P	P	NP-complete
α-weighted-\$bribery	P	NP-complete	NP-complete

Table 1: The complexity of bribery within scoring protocols.

elements, if we look at the image of only a certain restriction of the partition problem we can modify the thus-obtained elections to obey the desired conditions. Finally, we show that the restricted partition problem used above is NP-hard. This completes the sketch of the proof of Theorem 4.4.

In the beginning of this section we noted that approval voting is an example of an election system where bribery is NP-complete, whereas manipulation is easy. We mention that bribery in approval elections is actually very easy, provided that one looks at a slightly different model. Our bribery problems allow us to completely modify the approval vector of a voter. This may, however, be too demanding since a voter might be willing to change some of his or her approval vector's entries but not to completely change his or her approval vector. In particular, in the approval-bribery′ problem we will ask whether it is possible to make our favorite candidate p a winner by at most k entry changes in the approval vectors. We also define the weighted and priced versions of approval-bribery′ in the natural way.

Theorem 4.6 approval-bribery′, approval-\$bribery′, approval-weighted$_{\text{unary}}$-\$bribery′ *and* approval-weighted-\$bribery$'_{\text{unary}}$ *are in P.* approval-weighted-\$bribery′ *is NP-complete.*

Which of the bribery models for approval is more practical depends on the setting. For example, bribery′ seems more natural when we look at the web and treat web pages as voting by linking to other pages. It certainly is easier to ask a webmaster to add/remove a link than to completely redesign the page.

5 Succinct Elections

So far we have discussed only nonsuccinct elections—ones where voters with the same preference lists (and weights, if voters are weighted) are given by listing them one at a time (as if given a stack of ballots). It is also very natural to consider the case where each preference list has its frequency conveyed via a count (in binary), and we will refer to this as "succinct" input. Succinct in curly braces within a name of a bribery problem will describe the fact that it holds in both cases, e.g., if we say that plurality-{succinct}-bribery is in P, we mean that both plurality-bribery and plurality-succinct-bribery are in P. (By the way, Theorem 3.1, by a similar but more careful algorithm than the one mentioned right after it, also holds for the succinct case.)

In this section we provide P membership results regarding succinctly represented elections with a fixed number of candidates. (Such results for the case of succinct representation immediately yield results for the nonsuccinct case.) The most useful tool here is Lenstra's [Len83] extremely powerful result that the integer programming feasibility problem is in P when the number of variables is bounded. Lenstra's algorithm has a very large constant factor in its running time, but what we are after are P-membership results and tools for obtaining them and not actual optimized algorithms.

Using the integer programming approach we obtain polynomial-time algorithms for bribery under scoring protocols in both the succinct and the nonsuccinct cases. The same approach yields a similar result for manipulation. (The nonsuccinct case for manipulation was already obtained by Conitzer and Sandholm [CS02a].)

Theorem 5.1 *For every scoring protocol $\alpha = (\alpha_1, \ldots, \alpha_m)$, both* α-{succinct}-bribery *and* α-{succinct}-manipulation *are in P.*

The power of the integer programming approach is not limited to the case of scoring protocols. In fact, the seminal paper of Bartholdi, Tovey, and Trick [BTT89b] shows that applying this method to computing the Dodgson score in nonsuccinct elections with a fixed number of candidates yields a polynomial-time score algorithm (and though they did not address the issue of succinct elections, one can see that there too this method works perfectly). Applying an integer programming attack for the case of bribery is a bit more complicated, since one has both the issue of the bribes and the issue of the exchanges involved in computing the Dodgson scores. But even in this setting one can represent the question as an integer programming feasibility problem, and thus via Lenstra's algorithm we have the following result.

Theorem 5.2 *For each fixed number of candidates,* DodgsonScore-{succinct}-bribery *is in P when restricted to that number of candidates.*

By this we mean that in polynomial time we can test if a given bribe suffices to obtain or beat a given Dodgson score for our favored candidate (the Dodgson score of candidate c is the number of switches needed to be done in voters' preference lists to make c the Condorcet winner). Using binary search we may compute the minimum bribe needed to make our favored candidate have a given Dodgson score.

In Young elections ([You77]; see also [RSV03], which proves that the winner problem in Young elections is complete for parallel access to NP) the score of a candidate is the number of voters that need to be removed to make that candidate a Condorcet winner.

Theorem 5.3 *For each fixed number of candidates,* YoungScore-{succinct}-bribery *is in P when restricted to that number of candidates.*

The issue of actually making a candidate p a winner (a unique winner, if we are studying the unique winner case) of Dodgson elections is, as already indicated, much more difficult and a direct attack using integer linear programming seems to fail. Nonetheless, combining the integer programming method with a brute-force algorithm resolves the issue for the nonsuccinct case.

Theorem 5.4 *For each fixed number of candidates,* Dodgson-bribery*,* Dodgson-\$bribery*,* Young-bribery*, and* Young-\$bribery *are all in P.*

On the other hand, using integer programming, we obtain polynomial-time algorithms for bribery in Kemeny elections in both the succinct and nonsuccinct cases.

Theorem 5.5 *For each fixed number of candidates,* Kemeny-{succinct}-bribery *is in P when restricted to that number of candidates.*

In brief, Kemeny's system elects each candidate who is most preferred in at least one preference order that maximizes the number of agreements with the voters' preferences, where for two candidates, a and b, two preference orders agree if they both place a ahead of b or both place b ahead of a.

6 Research Directions

This paper provides a detailed study of the complexity of bribery with respect to plurality rule and, more generally, scoring protocols. This paper also provides tools and results regarding many other election systems such as approval voting and Dodgson elections.

There are several directions in which further research on bribery might go. The most obvious one is to study the complexity of bribery for other election systems. Another very interesting route is to study approximation algorithms for \$bribery problems. It would also be interesting to study the complexity of bribery in other settings, such as with incomplete information, multiple competing bribers, or more complicated bribe structures.

Acknowledgments: We are very grateful to Samir Khuller for helpful conversations about the Bartholdi, Tovey, and Trick integer programming attack on fixed-candidate Dodgson elections, and we thank the anonymous AAAI-06 and COMSOC-06 referees for helpful comments.

References

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

[BTT89b] J. Bartholdi, III, C. Tovey, and M. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165, 1989.

[BTT92] 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.

[CLS03] V. Conitzer, J. Lang, and T. Sandholm. How many candidates are needed to make elections hard to manipulate? In *Proceedings of the 9th Conference on Theoretical Aspects of Rationality and Knowledge*, pages 201–214. ACM Press, July 2003.

[CS02a] V. Conitzer and T. Sandholm. Complexity of manipulating elections with few candidates. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pages 314–319. AAAI Press, July/August 2002.

[CS02b] V. Conitzer and T. Sandholm. Vote elicitation: Complexity and strategy-proofness. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pages 392–397. AAAI Press, July/August 2002.

[CS03] V. Conitzer and T. Sandholm. Universal voting protocol tweaks to make manipulation hard. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 781–788. Morgan Kaufmann, August 2003.

[DKNS01] 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, March 2001.

[Dod76] C. Dodgson. A method of taking votes on more than two issues. Pamphlet printed by the Clarendon Press, Oxford, and headed “not yet published”, 1876.

[DS00] J. Duggan and T. Schwartz. Strategic manipulability without resoluteness or shared beliefs: Gibbard–Satterthwaite generalized. *Social Choice and Welfare*, 17(1):85–93, 2000.

[EL05] E. Elkind and H. Lipmaa. Small coalitions cannot manipulate voting. In *Proceedings of the 9th International Conference on Financial Cryptography and Data Security*, pages 285–297. Springer-Verlag *Lecture Notes in Computer Science #3570*, 2005.

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

[FHH06] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. How hard is bribery in elections? Technical Report TR-895, Department of Computer Science, University of Rochester, Rochester, NY, April 2006. Revised, September 2006.

[Gib73] A. Gibbard. Manipulation of voting schemes. *Econometrica*, 41(4):587–601, 1973.

[HH05] E. Hemaspaandra and L. Hemaspaandra. Dichotomy for voting systems. Technical Report TR-861, Department of Computer Science, University of Rochester, Rochester, NY, April 2005. Journal version to appear in *Journal of Computer and System Sciences.*

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

[Len83] H. Lenstra, Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

[LLS75] R. Ladner, N. Lynch, and A. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1(2):103–124, 1975.

[PR06] A. Procaccia and J. Rosenschein. Junta distributions and the average-case complexity of manipulating elections. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 497–504. ACM Press, May 2006.

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

[Sat75] M. Satterthwaite. Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10(2):187–217, 1975.

[You77] H. Young. Extending Condorcet's rule. *Journal of Economic Theory*, 16(2):335–353, 1977.

Piotr Faliszewski
Department of Computer Science, University of Rochester
Rochester, NY 14627 USA, www.cs.rochester.edu/u/pfali

Edith Hemaspaandra
Department of Computer Science, Rochester Institute of Technology
Rochester, NY 14623 USA, www.cs.rit.edu/∼eh

Lane A. Hemaspaandra
Department of Computer Science, University of Rochester
Rochester, NY 14627 USA, www.cs.rochester.edu/u/lane