Search versus Decision for Election Manipulation Problems*

Edith Hemaspaandra[1], Lane A. Hemaspaandra[2], and Curtis Menton[2]

1 **Department of Computer Science, RIT, Rochester, NY 14623, USA**
2 **Dept. of Computer Science, Univ. of Rochester, Rochester, NY 14627, USA**

Abstract

Most theoretical definitions about the complexity of manipulating elections focus on the decision problem of recognizing which instances can be successfully manipulated, rather than the search problem of finding the successful manipulative actions. Since the latter is a far more natural goal for manipulators, that definitional focus may be misguided if these two complexities can differ. Our main result is that they probably do differ: If integer factoring is hard, then for election manipulation, election bribery, and some types of election control, there are election systems for which recognizing which instances can be successfully manipulated is in polynomial time but producing the successful manipulations cannot be done in polynomial time.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes; F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Search vs. decision; application of structural complexity theory

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.377

1 Introduction

Elections are such a ubiquitous model for human and electronic collective decision-making—and during the past few decades, with the rise of computers, multiagent systems, and the internet, elections have become important even in many “modern” challenges such as collaborative filtering/recommender systems, planning, and reducing web spam—that much work has been devoted to studying how to manipulate elections. However, the broad stream of theoretical work on the computational complexity of manipulative attacks on elections (see the surveys [15, 12]) is largely centered on the complexity of the decision versions: Given an instance, determining whether there exists a successful manipulation (typically, ensuring that a given candidate wins, or ensuring that a given candidate does not win) of the given sort.

As a running example that we will use in this introduction, consider unweighted noncoalition (i.e., a single manipulator) manipulation, which was central in one of the seminal papers on manipulation ([1], see also [2]). For this problem, relative to some fixed election system, the inputs are the candidate set, the voter set consisting of a collection of nonmanipulative voters (whose preferences are each typically expressed by each voter as a preference ballot, e.g., Gore > Nader > Bush), and a single manipulative voter who has not yet set her vote but who has a “preferred” candidate p. And the question is: Does there exist a preference

* Supported by grants NSF-CCF-{0915792,1101452,1101479} and ARC-DP110101792, and two Humboldt Foundation Bessel Research Awards. Work done in part while visiting the University of Düsseldorf.

(vote) the manipulative voter can cast that will make p win the election? This is typically viewed as a decision (language) problem, namely, as the set of all instances for which the answer to that question is "Yes."

Of course, what a manipulator might most want is not to know a successful manipulation *exists* (a decision problem), but rather to know what *specific action* (what vote, bribe, etc.) to take to achieve success (a search problem). For the case of our unweighted noncoalition manipulation example, the search version would be a function that takes the same input as the decision version but then either outputs that no successful strategic vote for the manipulative voter exists or, if a successful vote does exist, *outputs a successful vote—one that makes p win.*

This paper studies whether these two goals' achievability can differ: whether decision versions of election problems can be easy yet their search versions intractable.

Virtually all papers in this area, to prove polynomial-time results for deciding when manipulative actions can succeed, actually give polynomial-time algorithms to produce the successful action. So one might suspect that perhaps that is always the case. For manipulation, bribery, and some types of control, we prove otherwise, under a complexity-theoretic hypothesis that is widely believed true. Our main contributions are:

- If $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$, then for each of manipulation (including in particular the case of our running example, unweighted noncoalition manipulation), bribery, and certain types of partition-control, there exist election systems for which there are polynomial-time algorithms to determine whether each given instance has a successful manipulative action, but no polynomial-time algorithm can exist that given an instance that is manipulable provides the successful manipulation. (It is widely believed in cryptography that integer factoring is hard. It is well known that if integer factoring is hard then $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$.) Informally put, the situation is that the frustrated world of polynomial-time computation will have to say things such as, "I can totally guarantee you that there are strategic votes you can cast to make Barack Obama win in the given electoral setting, but I have no idea what those votes are." We show that this bizarre setting can even occur in extremely simple cases, such as unweighted noncoalition (i.e., where we have just a single manipulative voter) manipulation. It follows immediately from our results that if $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ then for each of the above-mentioned manipulative actions there exists an election system in which the search problem does not polynomial-time Turing reduce to the decision problem.

  To the best of our knowledge, this is the first result separating, even conditionally, search from decision in the setting of computational social choice (see [7, 26])—an area whose core definitions on election manipulative-actions, which date back twenty years, are framed in terms of decision problems.

  To the best of our knowledge, our proof is the first time the complexity-theoretic Borodin-Demers Theorem, from the 1970s, has found application in an applied domain.
- In contrast, we show that for all the standard types of election-control actions based on adding or deleting voters or candidates, and for some of the standard election-control actions based on partitioning, the search problem (finding how to succeed) polynomial-time Turing reduces to the decision problem (knowing when one can succeed). It follows that, for these manipulative actions and for every election system, the bizarre type of behavior mentioned earlier cannot occur: Easy recognition of instances where success is possible implies polynomial-time algorithms for how to achieve success.

  While proving this, we notice that two pairs of control attacks assumed to differ in fact are identical problems, namely, for every election system, destructive control by partition of candidates and destructive control by run-off partition of candidates are the same set in both the standard tie-breaking cases (ties-eliminate and ties-promote); this reduces by two the number of distinct, standard control types. This is the first collapse of standard control types that we are aware of.

- Regarding the results of the first bullet point above, which, when $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ make decision easy but search hard, one might worry that search may be only infrequently hard. We address this by, as Theorem 9, constructing manipulative-action problems whose search versions are *just as often hard as are those problems in* $\mathrm{NP} \cap \mathrm{coNP}$ *that have the highest density of hardness*, give or take a slight degree of flexibility. These results provide a transference of density-of-hardness from a class to a particular type of concrete problem.

Given that $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ suffices to for some systems make the natural problem to care about (the search version) hard even as the problem that has been the theoretical literature's central definition (the decision version) declares the problem easy, we suggest that in definition and problem framing it may now be good to more energetically stress the importance of the search versions of election manipulation problems.

2 Preliminaries

An election will consist of a set C of candidates and a (multi)set V of voters (who for us will be given just as their preferences). For all the cases discussed in this paper, each voter's preferences will be a tie-free linear ordering of the candidates. We assume each vote is input distinctly (i.e., the voters' preferences come in as separate ballots; but it would be cheating for us to use the order of that input list within our proofs). So a typical election might be $C = \{\mathrm{Alice}, \mathrm{Bob}, \mathrm{Carol}\}$ and the voter (multi)set might be $V = \{(\mathrm{Carol} > \mathrm{Bob} > \mathrm{Alice}), (\mathrm{Bob} > \mathrm{Alice} > \mathrm{Carol})\}$. Most familiar election systems, such as plurality-rule elections, don't care about voter names; our constructions never need to use voter names, and so like most papers we don't have voter names in V.

Election systems, or voting systems, map from an election instance (C, V) to a set of winners (i.e., to a set W, $\emptyset \subseteq W \subseteq C$). (Pure social choice papers often definitionally exclude the case $W = \emptyset$, but like most papers on computational social choice we allow it.)

For each fixed election system $\mathcal{E}$, one can define the election winner problem as follows (see [3]).

Name: $\mathcal{E}$-winner, or the winner problem for $\mathcal{E}$.
Given: Election (C, V) and candidate $p \in C$.
Question: Is p a winner of the election (C, V) under election system $\mathcal{E}$?

This is actually, in the way universally accepted in computer science, describing a set, i.e., a language. That set is the set of all triples $\langle C, V, p\rangle$ such that the answer to the question is "Yes."

We now briefly present the key definitions for the three most commonly studied types of manipulative actions: manipulation, bribery, and control. These three types were first studied, respectively, by Bartholdi, Orlin, Tovey, and Trick [2,1], Faliszewski, Hemaspaandra, and Hemaspaandra [11], and Bartholdi, Tovey, and Trick [4] for the "constructive" cases, i.e., where the goal is to make a particular candidate be a winner.[1] The "destructive" cases,

[1] We say "be a winner" as this entire paper will focus on that notion, known as the nonunique-winner

where the goal is to ensure that a particular candidate is not a winner, were introduced by Conitzer, Sandholm, and Lang [8] for manipulation, by Faliszewski, Hemaspaandra and Hemaspaandra for [11] for bribery, and by Hemaspaandra, Hemaspaandra and Rothe [20] for control.

The manipulation problem is defined as follows, and models whether a coalition of strategic voters can make a certain candidate win.

Name: $\mathcal{E}$-unweighted coalition manipulation, or the unweighted coalition manipulation problem for $\mathcal{E}$; for short, the manipulation problem for $\mathcal{E}$.

Given: Candidate set C, nonmanipulative voter set V_1 (as a collection of preference ballots each with preferences over C), manipulative voter set V_2 (since we don't have names, this will be input as a unary string, 1^k, to indicate the number, k, of manipulative voters), and a candidate $p \in C$.

Question: Is there some choice of preferences for the manipulative voters such that p is a winner in the election in system $\mathcal{E}$ with candidates C and with both the nonmanipulative and the manipulative voters voting?

This again is a decision problem consisting of the set of all inputs yielding the answer "Yes." However, there is a very natural search problem associated with this, which we will call manipulation search, i.e., finding the successful action. In particular, a function f solves the manipulation search problem (for a given election system) if on all inputs where the Question's answer is "No" (i.e., all inputs not in the set that is the decision version) the function indicates in some clear way (e.g., by outputting -1) that manipulation is not possible, and on each input that belongs to the decision version, f specifies settings to the preferences of the manipulative voters in such a way that those result in p being a winner in the election $(C, V_1 \cup V_2)$. If some solution for the manipulation search problem is a polynomial-time computable function, we will say that the manipulation search problem is polynomial-time computable.

One can also define "weighted" coalition manipulation, where each manipulative and nonmanipulative voter has a weight (how many times her vote counts). Our results on manipulation all will hold for that case too. But it is more interesting that the results hold even in the unweighted case—and indeed, our proofs establish that they hold even when the number of manipulative voters is limited to being at most one.

Unlike manipulation, in bribery all voters have initial preferences. In the simplest model of bribery, voters are unweighted and each has unit cost to bribe. (By varying these parameters, [11] obtained three other models: unweighted, priced; weighted, unpriced; and weighted, priced. Our results on bribery hold in all four models.) This problem models whether having the ability to reshape (bribe) the preferences of a number of voters allows one to make a given candidate win.

Name: $\mathcal{E}$-bribery, or the bribery problem for $\mathcal{E}$.

Given: Election (C, V), candidate $p \in C$, and integer $b \geq 0$.

Question: Does there exist some collection of at most b voters, and a way of setting their votes, so that in the election under $\mathcal{E}$ in which those votes are thus set and the other voters cast the votes the input specified for them, p is a winner?

model (i.e., allowing ties). That model has broadly been the one previous papers favored (except the earliest work on control, but we feel that for control too this model is the more natural one).

Again, this is and should be viewed as a decision problem—as a set. It has the natural search version, which we will call bribery search.

Finally, we come to election control, the most varied, the most difficult to describe, but in our opinion the most interesting of the three most studied types of manipulative attacks on elections. Control asks whether by various adjustments to the participation and structure of an election, a given candidate can be made a winner. A natural set of control actions was specified in [4], the seminal paper on control, and we adopt that set, very slightly modified—as is now done in most papers—to treat adding of candidates symmetrically with the other add/delete types (as suggested by [14]) and to be clear in the "partition" cases about how first-round ties are handled (following [20]). Those control types are adding candidates, deleting candidates, adding voters, deleting voters, partition of voters, run-off partition of candidates, and partition of candidates. These loosely model many real-life settings, ranging from get-out-the-vote drives, to voter suppression, to having a culling "primary" round, to encouraging (or discouraging) "spoiler" candidates (see [14] for discussion of how these model various real-life scenarios). Each of the three partition control types is two control types—one (denoted by a TP—"ties promote"—modifier) for the model in which if a first-round election has multiple winners they all move forward to the second round, and one (denoted by a TE—"ties eliminate"—modifier) for the model in which one moves forward from a first-round election only if one is the unique winner of that contest.

Due to space limitations, we define here only the control type for which we include a proof sketch of our main result. The other control types are each defined in the intuitively natural way, and their full definitions can be found in the TR version [19], *which also contains proofs of all our results.*

▶ **Definition 1.** Let $\mathcal{E}$ be an election system. In the control by run-off partition of candidates problem for $\mathcal{E}$, in the TP or TE tie-handling rule model, we are given an election (C, V) and a candidate $p \in C$. Is there a partition of C into C_1 and C_2 such that p is a winner of the two-stage election where the winners of subelection (C_1, V) that survive the tie-handling rule compete against the winners of subelection (C_2, V) that survive the tie-handling rule? Each subelection (in both stages) is conducted using election system $\mathcal{E}$.[2]

Control problems are decision problems, i.e., sets. And they have the obvious search versions, which we will refer to in ways analogous to those we mentioned earlier regarding manipulation.

All the manipulation, bribery, and control problems defined so far are about trying to make a certain candidate be a winner. We will henceforward when mentioning these problems always add the word "constructive," to indicate that the problem is about making the specified candidate be a winner. As alluded to earlier, for every problem we have defined there is a "destructive" version, where the question is whether one can ensure that the specified candidate is not a winner. Both the constructive and destructive problems have both decision and search versions, in the obvious way.

A (decision or search) problem A is said to polynomial-time Turing reduce ($\leq_T^p$-reduce) to a decision problem B if there is a machine M such that (a) M^B runs in polynomial time (relative to the length of its input), and (b) if A is a decision problem then the language

[2] When speaking of an election, (C', V'), we always implicitly mean that each vote in V' is passed to the election system only as the version of itself restricted to the candidates in C'. This is the normal approach in defining control types, but we stress it because if we did not follow this approach, we might cheat in some of our constructions and use parts of a vote regarding candidates not in the election to pass/control information.

Table 1 Results summary. Key: "S $\leq$ D" is shorthand for: For each election system $\mathcal{E}$, the named constructive or destructive manipulative action has the property that its search version polynomial-time Turing reduces to its decision version. (Note that this implies that it is impossible for its decision version to be polynomial-time computable but its search version not to be polynomial-time computable.) "S $\not\leq$ D" is shorthand for: If P $\neq$ NP $\cap$ coNP, then there exists an election system $\mathcal{E}$, having a polynomial-time winner problem, such that the named constructive or destructive manipulative action's decision problem is in polynomial time but its search problem is not in polynomial time. (Note that this implies that if P $\neq$ NP $\cap$ coNP, then there is an election system $\mathcal{E}$ such that for the named constructive or destructive manipulative action, search does not polynomial-time Turing reduce to decision.)

Manipulative Action	Constructive	Destructive
bribery	S $\not\leq$ D	S $\not\leq$ D
control by adding voters	S $\leq$ D	S $\leq$ D
control by deleting voters	S $\leq$ D	S $\leq$ D
control by partition of voters, ties promote	S $\not\leq$ D	S $\not\leq$ D
control by partition of voters, ties eliminate	S $\not\leq$ D	S $\not\leq$ D
control by adding candidates	S $\leq$ D	S $\leq$ D
control by deleting candidates	S $\leq$ D	S $\leq$ D
control by partition of candidates, ties promote	S $\not\leq$ D	S $\leq$ D
control by partition of candidates, ties eliminate	S $\not\leq$ D	S $\leq$ D
control by run-off partition of candidates, ties promote	S $\not\leq$ D	S $\leq$ D
control by run-off partition of candidates, ties eliminate	S $\not\leq$ D	S $\leq$ D
control by unlimited adding of candidates	S $\leq$ D	S $\leq$ D
manipulation	S $\not\leq$ D	S $\not\leq$ D

accepted by M^B is A, and if A is a search problem then M^B computes a function that is a solution of the search problem (M^B means machine M given a unit-cost subroutine testing membership in B); this is the standard definition of polynomial-time Turing reductions, which along with polynomial-time many-one reductions are the central ways computer science links and compares the complexity of problems. For example, if we say that $\mathcal{E}$-manipulation search polynomial-time Turing reduces to $\mathcal{E}$-manipulation, that means that given an instance of the $\mathcal{E}$-manipulation problem (but being interested in getting an action, i.e., we are doing the search version), we can in polynomial time, given access to an oracle for the set $\mathcal{E}$-manipulation, correctly either state that successful manipulation is impossible or output a successful manipulation. We move directly on to the presentation of our results, and then provide a discussion of related work.

3 Results

The tightly related goals of this paper are to determine for which manipulative actions

(a) for all election systems, search (polynomial-time Turing) reduces to decision,

and to determine for which manipulative actions

(b) there exists some election system $\mathcal{E}$, whose winner problem is in P, for which the decision version of the manipulative action is in P yet the search version of the decision problem is not polynomial-time computable (i.e., no polynomial-time function solves the search version).

These are related, as "(a)" implies "NOT (b)."

For manipulation, bribery, and every standard type of control, we in effect strongly resolve this. That is, for some, we prove (a)—which of course implies NOT (b) (in fact, it implies even that "NOT (b′)," where (b′) is (b) with the "winner problem in P" requirement removed). And for all the others we prove, under the complexity-theoretic assumption $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$, that (b) holds—which of course implies NOT (a). The more striking group of cases is the latter collection—manipulative actions for which for some election system with an easy (i.e., polynomial-time) winner problem we can easily (i.e., in polynomial time) for a given setting determine whether a successful attack exists, and yet there can exist no polynomial-time algorithm to always tell us what the successful attack action (that we know exists!) is.

In the process of proving the latter group of cases we will do even more than promised above. We will not only show that $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ implies (b), but we also will characterize (b), for each of those manipulative actions, as being equivalent to the right-hand side condition of the so-called Borodin-Demers Theorem from computational complexity theory (i.e., the "then" part of Theorem 4 below). So, although we need a rich variety of complex election schemes and tricky coding schemes to prove our results, from those results and that work we establish that twelve different instances of whether (b) holds are, deep down, the same issue.

In the process of proving the other group of cases we will note that two pairs of control types that have always been viewed as distinct in fact pairwise collapse: viewed as sets, they are the exact same set. So all previous papers that gave separate proofs for the two elements of a collapsing pair were proving the same result twice. To be fair to earlier papers it is important to mention that of the two pairs that we show to collapse (in the nonunique winner model), only one of those pairs collapses in the unique winner model; that itself is also a new result.

3.1 Cases When the Manipulative-Action Decision Problem Is Easy but Its Search Problem Is Hard

Our main result, showing that if $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ then there are easy election systems (i.e., having a polynomial-time winner problem) whose manipulative-action decision problem is easy but whose manipulative-action search problem is hard, is the following.

▶ **Theorem 2.** *If $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$, then for each manipulative action $\mathcal{A}$ marked "$S \not\leq D$" in Table 1, there exists an election system $\mathcal{E}$ (which may differ based on $\mathcal{A}$), whose winner problem is in polynomial time, such that the $\mathcal{A}$-decision problem for $\mathcal{E}$ is in P but the $\mathcal{A}$-search problem for $\mathcal{E}$ is not polynomial-time computable.*

▶ **Corollary 3.** *If $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$, then for each of the manipulative actions $\mathcal{A}$ covered by Theorem 2, $\mathcal{A}$-search for $\mathcal{E}$ does not polynomial-time Turing reduce to $\mathcal{A}$-decision for $\mathcal{E}$.*

Let us present the idea behind the proof of Theorem 2, focusing in particular as an example on constructive control by run-off partition of candidates in the ties-promote model. So, let us use the statement's hypothesis, and assume that $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ holds. We invoke a complexity-theoretic result known as the Borodin-Demers Theorem. To the best of our knowledge, the Borodin-Demers Theorem has never before been applied in the study of elections, computational social choice, multiagent systems, or for that matter anywhere outside of computational complexity theory.

▶ **Theorem 4** ([6], see [18,25] for the form used here)**.** *If $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ then there is a set B so (1) $B \in \mathrm{P}$, (2) $B \subseteq \mathrm{SAT}$, and (3) no P machine can find solutions for all formulas in*

B (that is, for no *polynomial-time computable function g do we have $(\forall f)[f \in B \Rightarrow g(f)$ is a satisfying assignment of $f])$.*

So we have something quite striking: A set of boolean formulas that are easily recognized as being satisfiable but for which it is not in general easy to find how they can be satisfied, i.e., every polynomial-time machine fails on some of them (indeed, on infinitely many, as otherwise one could finitely patch). (The Borodin-Demers Theorem certainly does *not* say that if $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ then search does not reduce to decision for SAT; it is well-known that for SAT—and indeed for any NP-complete problem—search $\leq_T^p$-reduces to decision. However, we will use Borodin-Demers as a tool to show that in certain election settings search does not reduce to decision if $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$.) Our goal, of course, is to shoehorn the set B into the world of election manipulation for a variety of manipulative actions. Of course, each manipulative action comes with its own form and definition, and so for many such shoehorning is essentially impossible—as we show in Section 3.2. But for others, we can do this, sometimes smoothly and sometimes through extreme, difficult contortions. The difficulty is that the structure of many electoral manipulations, and our goal to realize a separation with respect even to some election system with a polynomial-time winner problem, very much ties our hands. And in fact, even for our results here, the different manipulative actions have enormously differing proofs, as each proof must be tailored to the manipulative action.

Nonetheless, the general approach is clear and shared, although the implementations and constructions differ wildly. The general approach is given a set B from the Borodin-Demers result, we must build an election system $\mathcal{E}$, whose winner problem is in P, such that for our manipulative action the decision problem is in P but the search problem is not polynomial-time computable. To do this, our election system $\mathcal{E}$ will clearly need to be very much attuned to B. It typically will be interpreting voters, candidates, collections of voters, and collections of candidates as variously trying to specify a Borodin-Demers "puzzle"—i.e., an obviously satisfiable formula (a string $x \in B$), and it also will interpret some similar things about its input as trying to propose solutions to that puzzle.

To really explain how this works in practice would require going through the actual proofs (which we provide in [19]). But to give an idea of the flavor, let us speak here in a high-level, handwaving way about a specific example (that is neither our hardest nor our easiest case), namely constructive control by run-off partition of candidates in the ties-promote model.

Proof (sketch, for the just-mentioned case): Our scheme here is to hope—although other inputs won't trip us up—that our input consists of two almost-copies of a Borodin-Demers puzzle x, namely that part of our input is $x0$ and $x1$, $x \in B$. In particular, we'll hope that the lexicographically two smallest candidates have those strings as their names. Suppose that the obviously satisfiable formula x (for concreteness of this sketch) is 1000 bits long and has 27 variables. Then we will hope to have exactly $2 \cdot 27 = 54$ other candidates, who will all form a lexicographically contiguous segment starting at, say $0^{5 \cdot 1000}$, i.e., the first of the 54 candidates is named 0^{5000}, the second is named $0^{4999}1$, the third is named $0^{4998}10$, and so on. Now, we'll interpret these strings as 27 pairs—the first two, the next two, and so on. And we'll set up our election system so that it will try to ensure that exactly one of each pair goes on one side of the partition in any partition that will lead to victory of $x0$. The election system if it sees in its candidate set $x0$, $x \in B$, will compute the size and number of variables of x, will see if it has the right collection of other candidates to indicate it has precisely one from each of the 27 pairs, will then interpret the low-order bit of each of those pair-choices as the ith bit of a guessed satisfying assignment for x, and if that assignment does satisfy x, will make $x0$ the one and only winner. Also, the election system when its input contains $x1$, $x \in B$, will check that it also has precisely one candidate from each of the 27 pairs (and no candidates other than those and $x1$), and if so $x1$ and only $x1$ will win—it does not in this case do any satisfiability check. A third and final case in which we will have a winner is if the candidate set is $\{x0, x1\}$, $x \in B$, in which case $x0$ and only $x0$ will win. And these three cases are the only ways to win.

Now, recall that run-off partition splits the candidates into two groups for primary elections and then runs the winners of those against each other. If the input set is of just the dream-case form we have described, and we ask whether $x0$ can by run-off partition, ties-promote, be made a winner of the overall election, the answer is obviously "Yes," as $x \in B$ is satisfiable and so the partition that puts into one side of the partition $x0$ and precisely a set of one-per-pair candidates encoding a satisfying assignment and puts the rest on the other side will have $x0$ win its first-round contest, will have $x1$ win its first-round contest, and will have $x0$ win the second-round contest between $x0$ and $x1$.

But it is possible to see that if we have a polynomial-time algorithm for the search problem of how to make $x0$ win, that on the special input we just described, any search-problem output, i.e., any successful partition, will immediately make clear a satisfying assignment of x, as the election system in fact will force that. So if we had a search-problem polynomial-time algorithm, the third property (the one about no FP function always yielding solutions) of the Borodin-Demers set B would be violated. So search for our election system is not polynomial-time computable.

But our election system clearly does have a P winner problem—it is just three simple cases to check. So all that remains is to show that the decision problem for this control type is in P. Note that we need a P algorithm that works for *all* inputs—not just inputs so nice as to have our dream-case format. However, when one carefully checks everything, with the system very clearly specified, one can see that this holds also (see our full version of this paper [19]). This is the part that causes a large part of the complexity of the election system; for example, the simpler system without the $x1$ requirement will fail this requirement. ❑

Now, Theorem 2 gives twelve cases where $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ implies the existence of a P-winner problem election system where for a particular manipulative action decision is easy but search is hard. It is natural to wonder whether the converses of some or all of these twelve results hold. We note that either all of the converses hold or none do, and which of those cases holds is identical to a long-open issue in complexity theory, namely, whether the converse of the Borodin-Demers Theorem holds. Let us call the three-part right-hand side of the Borodin-Demers Theorem the "Borodin-Demers Condition." We claim the following result.

▶ **Theorem 5.** *For each of the twelve manipulative actions $\mathcal{A}$ referred to in Theorem 2, the following two conditions are equivalent:*

- *The Borodin-Demers Condition holds.*
- *There exists an election system $\mathcal{E}$, with a polynomial-time winner problem, such that the $\mathcal{A}$ decision problem for $\mathcal{E}$ is in* P *but the $\mathcal{A}$ search problem for $\mathcal{E}$ is not polynomial-time computable.*

Finally, one might worry, given the broad interest recently in how often NP-hard election manipulation problems are hard (e.g., [17]), that although Theorem 2's conclusion says that decision is easy while search is hard, the hardness for search that it speaks of is a worst-case notion of hardness, and so perhaps the hard instances form a very sparse set. This is a natural worry, but to partially address it we will as Theorem 9 prove that if even one set A in $\mathrm{NP} \cap \mathrm{coNP}$ is frequently hard, then all of our search cases are (in a certain sense) almost as frequently hard as that set A.

3.2 Cases Where Search Reduces to Decision

This section's main result states that for many manipulative actions search polynomial-time Turing reduces to decision.

▶ **Theorem 6.** *For each manipulative action $\mathcal{A}$ marked "$S \leq D$" in Table 1, and for each election system $\mathcal{E}$, the $\mathcal{A}$ search problem for $\mathcal{E}$ polynomial-time Turing reduces to the $\mathcal{A}$ decision problem for $\mathcal{E}$.*

Thus the behavior displayed in Theorem 2 is impossible for all of the above manipulative actions, even if Theorem 2's "winner problem in P" requirement is dropped.

▶ **Corollary 7.** *For each of the manipulative actions $\mathcal{A}$ referred to in Theorem 6, for no election system $\mathcal{E}$ can it be the case that the $\mathcal{A}$ decision problem for $\mathcal{E}$ is in* P *but the $\mathcal{A}$ search problem for $\mathcal{E}$ is not polynomial-time computable.*

The proofs can be found in [19]. But we mention that important to establishing the four cases with the most interesting proofs, and an interesting result in its own right, is the following. We show that two pairs of control types which in previous papers have been assumed to be distinct, are in fact identical.

▶ **Theorem 8.**
1. DC-RPC-TP = DC-PC-TP *(i.e., viewed as decision problems, destructive control by run-off partition of candidates in the ties-promote model is* exactly *the same problem—the same set—as is destructive control by partition of candidates in the ties-promote model).*
2. DC-RPC-TE = DC-PC-TE*.*

4 Related Work, Frequency of Hardness, and Open Directions

Although to the best of our knowledge search versus decision has not previously been a focus area in the long line of work on the complexity of manipulative attacks, the detailed analysis of the complexity of attacks on particular systems has been a focus area. For example, detailed classifications of the complexities of constructive and destructive control actions on specific systems can be found in such work as [14,10,9,5,24]. These papers are about specific systems. In contrast, our "search reduces to decision" results hold for all systems. Our "if $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$" results on the other hand use that complexity-theoretic hypothesis to build specific systems that make decision easy while making search hard.

Existing papers that give polynomial-time attack algorithms against specific systems typically do so by (at least implicitly) finding a polynomial-time solution to the search problem. Probably the definition most related to the interests of this paper is the definition of "certifiably vulnerable" of Hemaspaandra, Hemaspaandra, and Rothe [20], which captures the notion of demanding that an attack provide a successful action when one exists. That paper actually adds an "optimality" twist to that notion, but subsequent papers (e.g., [13, 16]) when using the notion of certifiability take it to mean providing some successful action when one exists, rather than the "smallest in size/effort/cost" such action.

Our search reduces to decision results of course hold on all inputs. But our "$\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$"-induced results put decision versions in P while ensuring that their search versions are not polynomial-time computable. That latter part is a worst-case claim. However, by a detailed look at the properties of (and length-stretching in, and injectivity of reductions related to) the proofs of both the Borodin-Demers Theorem and Theorem 2, we can prove (see [19]) the following result, that says that *we can construct manipulative-action problems* *within Theorem 2 whose search versions are just as often hard as are those problems in* NP ∩ coNP *(such as, potentially, problems related to factoring) that have the highest density of hardness*, give or take an ϵ of flexibility. Speaking more broadly, although our paper speaks in terms of keeping search algorithms out of polynomial time, its proof infrastructure is enough to strongly address the issue of how often failure occurs—or at least to strongly link that to the open issue of how densely hard sets in NP ∩ coNP can be.

▶ **Theorem 9.** *If f is any nondecreasing function, and for some set $A \in \mathrm{NP} \cap \mathrm{coNP}$ it holds that every polynomial-time membership-in-A-testing algorithm errs, at infinitely many lengths n (respectively, at almost every length n), on at least $f(n)$ of the strings up to that length, then for each manipulative action (that appears in our $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ theorems) there will exist an $\epsilon > 0$ and an election system having a polynomial-time winner problem such that each search algorithm for that manipulative action with respect to that election system will err, at infinitely many lengths n (respectively, at almost every length n), on at least $f(n^\epsilon)$ of the strings up to that length, but the decision problem will be in* P*.*

As a concrete example, if some set in NP ∩ coNP causes, for some $\epsilon > 0$, 2^{n^ϵ} errors up to length n at infinitely many lengths, by each P algorithm, then in our theorems we can, for some $\tilde{\epsilon} > 0$, have our search problems for infinitely many lengths make each polynomial-time solver err $2^{n^{\tilde{\epsilon}}}$ times up to that length.

There is far too large a literature exploring the many aspects of search versus decision to cite it all here, but as an indication of how broad the literature is we mention a paper related to search versus decision as it interacts with parallelism [22] and a paper related to P-selectivity and self-reducibility [21]. Of course, the present paper is looking at concrete cases of search versus decision, in the context of manipulative actions on elections.

Does $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ hold? There are a number of problems that are known to belong to NP ∩ coNP yet that despite intense effort have not been shown to belong to P. Such problems include important questions about lattice problems, stochastic games, parity games, and factoring. Regarding factoring, it is well known that if $\mathrm{P} = \mathrm{NP} \cap \mathrm{coNP}$ then integer factoring is in polynomial time; this is to many people very strong evidence that $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ (see [23]). *Note that, thus, if one believes factoring is hard, then by our results one must also believe that search and decision differ in complexity for many types of manipulative attack. The natural lesson to draw is that in framing definitions and questions, heightened attention should in the future be given to search versions.*

The problems mentioned in the previous paragraph are relevant to the most pressing open direction: Can one find existing—or build new but still highly *natural*—election systems for which, for some of the attacks we've discussed, the decision problem is in P but the search problem seems not to be in P? Note that since decision reduces to search, system-attack pairs known to have poly-time search algorithms or NP-hard decision problems are not reasonable possibilities here. Rather, the most attractive approach would be to find or more likely build natural election systems whose definitions involve seemingly NP-intermediate problems, such as factoring, lattice problems, and graph isomorphism.

Acknowledgments We are grateful to the STACS 2013 referees for helpful comments.

References

1 J. Bartholdi, III and J. Orlin. Single transferable vote resists strategic voting. *Social Choice and Welfare*, 8(4):341–354, 1991.
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. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165, 1989.

4 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.

5 D. Baumeister, G. Erdélyi, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Computational aspects of approval voting. In *Handbook of Approval Voting*. Springer, 2010.

6 A. Borodin and A. Demers. Some comments on functional self-reducibility and the NP hierarchy. Technical Report TR 76-284, Dept. of Comp. Sci., Cornell Univ., July 1976.

7 Y. Chevaleyre, U. Endriss, J. Lang, and N. Maudet. A short introduction to computational social choice. In *Proc. of SOFSOM-07*, pages 51–69. Springer-Verlag *LNCS #4362*.

8 V. Conitzer, T. Sandholm, and J. Lang. When are elections with few candidates hard to manipulate? *JACM*, 54(3):1–33, 2007.

9 G. Erdélyi, L. Piras, and J. Rothe. The complexity of voter partition in Bucklin and fallback voting: Solving three open problems. In *Proc. of AAMAS-11*, pages 837–844.

10 G. Erdélyi and J. Rothe. Control complexity in fallback voting. In *CATS-10*, pages 39–48.

11 P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. How hard is bribery in elections? *JAIR*, 35:485–532, 2009.

12 P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. Using complexity to protect elections. *Communications of the ACM*, 53(11):74–82, 2010.

13 P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. Multimode control attacks on elections. *JAIR*, 40:305–351, 2011.

14 P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Llull and Copeland voting computationally resist bribery and constructive control. *JAIR*, 35:275–341, 2009.

15 P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. A richer understanding of the complexity of election systems. In S. Ravi and S. Shukla, editors, *Fundamental Problems in Computing*, pages 375–406. Springer, 2009.

16 P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. The shield that never was: Societies with single-peaked preferences are more open to manipulation and control. *Inf. and Comp.*, 209(2):89–107, 2011.

17 E. Friedgut, G. Kalai, and N. Nisan. Elections can be manipulated often. In *Proc. of FOCS-08*, pages 243–249.

18 L. Hemachandra. *Counting in Structural Complexity Theory*. PhD thesis, Cornell University, Ithaca, NY, 1987. Available as Cornell CSD Technical Report TR87-840.

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

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

21 E. Hemaspaandra, A. Naik, M. Ogihara, and A. Selman. P-selective sets and reducing search to decision vs. self-reducibility. *JCSS*, 53(2):194–209, 1996.

22 R. Karp, E. Upfal, and A. Wigderson. The complexity of parallel search. *JCSS*, 36(1):225–253, 1988.

23 S. Kintali. NP int. coNP. `kintali.wordpress.com/2010/06/06/np-intersect-conp/`.

24 C. Menton. Normalized range voting broadly resists control. *ToCS*. To appear.

25 J. Rothe. Complexity of certificates, heuristics, and counting types, with applications to cryptography and circuit theory. Habilitation, Friedrich-Schiller-Universität Jena, 1999.

26 J. Rothe, D. Baumeister, C. Lindner, and I. Rothe. *Einführung in Computational Social Choice*. Spektrum Akademischer Verlag, 2011.