8-2009

Guarantees for the Success Frequency of an Algorithm for Finding Dodgson-Election Winners

Christopher Homan
Rochester Institute of Technology

Lane A. Hemaspaandra
University of Rochester

Guarantees for the Success Frequency of an Algorithm for Finding Dodgson-Election Winners[*]

Christopher M. Homan[†]
Department of Computer Science
Rochester Institute of Technology
Rochester, NY 14623 USA

Lane A. Hemaspaandra[‡]
Department of Computer Science
University of Rochester
Rochester, NY 14627 USA

September 19, 2005; revised June 11, 2006 and June 23, 2007

Abstract

In the year 1876 the mathematician Charles Dodgson, who wrote fiction under the now more famous name of Lewis Carroll, devised a beautiful voting system that has long fascinated political scientists. However, determining the winner of a Dodgson election is known to be complete for the Θ_2^p level of the polynomial hierarchy. This implies that unless P = NP no polynomial-time solution to this problem exists, and unless the polynomial hierarchy collapses to NP the problem is not even in NP. Nonetheless, we prove that when the number of voters is much greater than the number of candidates—although the number of voters may still be polynomial in the number of candidates—a simple greedy algorithm very frequently finds the Dodgson winners in such a way that it "knows" that it has found them, and furthermore the algorithm never incorrectly declares a nonwinner to be a winner.

1 Introduction

Suppose that a group of individuals is voting among three candidates (or alternatives), call these candidates a, b, and c. Further suppose that four-ninths of the voters rank the candidates, in order of strictly increasing preference, (c, b, a), that three-ninths rank them (a, b, c), and that the remaining two-ninths rank them (a, c, b). So five-ninths of the voters prefer b to a and six-ninths of the voters prefer b to c. A candidate such as b, i.e., a candidate who when paired against each other candidate is preferred by a strict majority of the voters, is called a *Condorcet winner* [Con85].

[*]A preliminary version of this paper was presented at the 2006 MFCS conference [HH06] and the 2006 COMSOC workshop.

[†]URL: `http://www.cs.rit.edu/~cmh`.

[‡]URL: `http://www.cs.rochester.edu/u/lane`. Supported in part by grant NSF-CCF-0426761, the Alexander von Humboldt Foundation's TransCoop program, and a Friedrich Wilhelm Bessel Research Award. Work done in part while visiting Heinrich-Heine-Universität Düsseldorf.

A *voting system* maps from candidates and voters (each of whose preference order over the candidates is represented by a strict, linear order) to a winner set, which must be a subset of the candidate set. As Condorcet noted in the 1700s [Con85] it is quite possible that sometimes there is no Condorcet winner. The classic example is the following. Given a choice between a, b, and c, one-third of a group might rank (in order of strictly increasing preference) the candidates (a, b, c), another third might rank them (b, c, a), and the remaining third might rank them (c, a, b). So two-thirds of the voters prefer b to a, two-thirds of the voters prefer a to c, and two-thirds of the voters prefer c to b. Clearly, there is no Condorcet winner. A voting system is said to obey the *Condorcet criterion* if it selects the Condorcet winner whenever one exists.

The Condorcet criterion may seem so natural that one might at first glance fail to see that many of the most widely-used voting rules do not meet it. For instance, in the "4/9, 3/9, 2/9" example above, if the voters were to use plurality-rule—i.e., a candidate is the winner if he or she receives the most first-place votes—then the group would choose a. As a second example, if instead the group were to hold an election under instant-runoff voting, c would win and b would be the first candidate eliminated. In instant-runoff voting, if no candidate is listed as the most favorite by a majority of the voters then each candidate who is listed as the most favorite by the fewest voters (i.e., b in our example) is dropped from preference lists and the election is recomputed with the edited lists; the process ends when a majority winner emerges (or due to ties all candidates have been removed). Examples where election systems fail to meet the Condorcet criterion are hardly new. In the 1800's Nanson [Nan82] wrote on the fact that well-known voting systems, such as Borda's [Bor84] famous rank-based point system, fail to satisfy the Condorcet criterion.

In 1876 the mathematician Charles Dodgson, whose name when an author of fiction was the now famous pseudonym of Lewis Carroll, devised a voting system [Dod76] that has many lovely properties and meets the Condorcet criterion. In Dodgson's system, each voter strictly ranks (i.e., no ties allowed) all candidates in the election. If a Condorcet winner exists, he or she wins the Dodgson election. If no Condorcet winner exists, Dodgson's approach is to take as winners all candidates that are "closest" to being Condorcet winners, with closest being in terms of the fewest changes to the votes needed to make the candidate a Condorcet winner. We will in Section 2 describe what exactly Dodgson means by "fewest changes," but intuitively speaking, it is the smallest number of sequential switches between adjacent entries in the rankings the voters provide. It can thus be seen as a sort of "edit distance" [SK83].

Dodgson wrote about his voting system in a pamphlet on the conduct of elections [Dod76]. It is now regarded as a classic of social choice theory [Bla58,MU95]. Dodgson's system was a relatively early example of a system satisfying the Condorcet criterion—though, remarkably, the Catalan mystic Ramon Llull in the thirteenth century had already proposed a system that satisfies the Condorcet criterion and has many other strengths (see [HP01,FHHR07]).

Although Dodgson's system has many nice properties, it also poses computational problems that are serious enough to make it an impractical choice for a general voting system that is guaranteed to work efficiently in any setting: The problem of checking whether a certain number of changes suffices to make a given candidate the Condorcet winner is NP-complete [BTT89], and the problem of computing an overall winner, as well as the related problem of checking whether a given candidate is at least as close as another given candidate to being a Dodgson winner, is complete for Θ_2^p [HHR97]. The Θ_2^p level of the polynomial hierarchy can be defined as the class of problems solvable with polynomial-time parallel access to an NP oracle ([PZ83], see also [Hem89,Wag90]). More recent work has shown that some other important election systems are complete for Θ_2^p: Hemaspaandra, Spakowski, and Vogel [HSV05] have shown Θ_2^p-completeness for the winner problem in Kemeny elections, and Rothe, Spakowski, and Vogel [RSV03] have shown Θ_2^p-completeness for the winner problem in Young elections. The above complexity results about Dodgson elections show, quite dramatically, that unless the polynomial hierarchy collapses there is no efficient (i.e., polynomial-time) algorithm that is guaranteed to always determine the winners of a Dodgson election. Does this then mean that Dodgson's widely studied and highly regarded voting system is unusable?

It turns out that if a small degree of uncertainty is tolerated, then there is a simple, polynomial-time algorithm, `GreedyWinner` (the name's appropriateness will later become clear), that takes as input a Dodgson election and a candidate from the election and outputs an element in {"yes", "no"} $\times$ {"definitely", "maybe"}. The first component of the output is the algorithm's guess as to whether the input candidate was a winner of the input election. The second output component indicates the algorithm's confidence in its guess. Regarding the accuracy of `GreedyWinner`, we show that for each (election, candidate) pair, it holds that if `GreedyWinner` outputs "definitely" as its second output component, then its first output component correctly answers the question, "Is the input candidate a Dodgson winner of the input election?" (This is an example of what we call *self-knowing correctness*; Definition 3.1 will make this formal.) We also show that for each $m, n \in \mathbb{N}^+$ the probability that a Dodgson election E selected uniformly at random from all Dodgson elections having m candidates and n votes (i.e., all $(m!)^n$ Dodgson elections having m candidates and n votes have the same likelihood of being selected [1]) has the property that there exists at least one candidate c such that `GreedyWinner` on input (E, c) outputs "maybe" as its second output component is less than $2(m^2 - m)e^{\frac{-n}{8m^2}}$.

Thus for elections where the number of voters greatly exceeds the number of candidates (though the former could still be within a (superquadratic) polynomial of the latter, consistently with the success probability for a family of election draws thus-related in voter-candidate cardinality going asymptotically to 1), if one randomly chooses an election $E = (C, V)$, then with high likelihood it will hold that for each $c \in C$ the efficient algorithm `GreedyWinner` when run on input (C, V, c) correctly determines whether c is a Dodgson

[1]Since Dodgson voting is not sensitive to the *names* of candidates, we will throughout this paper always tacitly assume that all m-candidate elections have the fixed candidate set $1, 2, \ldots, m$ (though in some examples we for clarity will use other names, such as a, b, c, and d). So, though we to be consistent with earlier papers on Dodgson elections allow the candidate set "C" to be part of the input, in fact we view this as being instantly coerced into the candidate set $1, 2, \ldots, m$. And we similarly view voter *names* as uninteresting.

winner of E, and moreover will "know" that it got those answers right. So `GreedyWinner` is a self-knowingly correct heuristic, and we will prove the above claims showing it is frequently (self-knowingly) correct.[2] Though the `GreedyWinner` algorithm on its surface is about *recognizing* Dodgson winners, as discussed in Section 3 our algorithm can be easily modified into one that is about, given an $E = (C, V)$, *finding* the complete set of Dodgson Winners and that does so in a way that is, in essentially the same high frequency as for `GreedyWinner`, self-knowingly correct. Later in this paper, we will introduce another frequently self-knowingly correct heuristic, called `GreedyScore`, for calculating the Dodgson score of a given candidate.

The study of greedy algorithms has an extensive history (see the textbook [CLRS01] and the references therein). Much is known in terms of settings where a greedy algorithm provides a polynomial-time approximation, for example [ACP95], and of guarantees that a greedy algorithm will frequently solve a problem—for example Kaporis, Kirousis, and Lalas study how frequently a greedy algorithm finds a satisfying assignment to a boolean formula [KKL02], and many additional excellent examples exist, e.g., [CK76,PT83,Sla97, Bro01]. However, each of these differs from our work in either not being about *self-knowing* correctness or, if about self-knowing correctness, in being about an NP-certificate type of problem. In contrast, as discussed in more detail immediately after Definition 3.1, the problem studied in this paper involves objects that are computationally more demanding than mere certificates for NP set membership.

Among earlier algorithms having a self-knowingly correct flavor, a particularly interesting one (though, admittedly, it is about finding certificates for NP set membership) is due to Goldberg and Marchetti-Spaccamela [GMS84]. Goldberg and Marchetti-Spaccamela construct for every $\epsilon > 0$ a modified greedy algorithm that is deterministic, runs in polynomial time (where the polynomial depends on ϵ), and with probability at least $1 - \epsilon$ self-knowingly finds an optimal solution of a randomly chosen (from a particular, but natural, distribution) instance of the knapsack problem. (They call their algorithm "self checking" rather than "self-knowingly correct.") Besides being about an NP-type problem rather than a "Θ_2^p"-type problem such as Dodgson elections, a second way in which this differs from our work is that the running time of our algorithms does not depend on the desired likelihood of correctness.

The concept of a heuristic that is effective on a significant portion of the problem instances of some very hard (e.g., not even in NP unless NP = coNP) problem and furthermore has the additional property that it can very frequently guarantee that its answers are correct is related to other theoretical frameworks. Parameterized complexity [DF99] studies hard problems that have efficient solutions when instances of the problem are restricted by fixing one or more of the parameters that describe the instances. The two most natural parameters of Dodgson elections are the number of candidates and the number of voters. It is known that with either of these parameters bounded by a constant, Dodgson elections

[2]As we just did in the second half of that sentence, in a slight abuse of notation we will sometimes speak of inputs on which a self-knowingly correct algorithm has the second component "definitely" as instances on which it is self-knowingly correct.

have polynomial-time algorithms [BTT89]. However, we are interested in cases when no such dramatic bounding of a parameter by a constant occurs.

Like average-case ([Lev86], see also the excellent surveys [Wan97a, Wan97b]) and smoothed [ST04] analyses, our analysis of `DodgsonWinner` is probabilistic. But while those methods are concerned with the expected value over segments of the problem domain of some resource of interest, typically time or a transformed version of time, we focus on a quite different property: correctness. Other recent papers also focusing on frequency (or probability weight) of correctness for hard voting-related problems (though not Dodgson voting) include work of Conitzer and Sandholm [CS06] and Procaccia and J. Rosenschein [PR07]. Those papers both assume a skewed underlying preference distribution, and in contrast our paper is focused on a uniform preference distribution.[3]

The paper that is closest to the present paper is the independent work of McCabe-Dansted, Pritchard, and Slinko [MPS06]. The core insight there is the same as our core insight: That for appropriate numbers of candidates and voters it will hold that, if voter preferences are independent and uniform, then on a very large portion of cases one can, purely by making swaps between one's candidate of interest and people adjacently beating it within votes, make the candidate a Condorcet winner (i.e., shift enough to precisely eliminate any shortfalls). We now in the main text summarize the three major differences between [MPS06] and our work. The footnotes to this summary contain additional discussion that, while important for completeness, may be safely omitted on a first reading.

The first major difference between our work and that of McCabe-Dansted, Pritchard, and Slinko is that their heuristic, DodgsonQuick, outputs just a score. Although that score is correct on a large portion of inputs, their DodgsonQuick doesn't on any particular input ever guarantee that the score is correct. In contrast, our `GreedyScore` heuristic is frequently *self-knowingly* correct—on a large portion of inputs, it gives the right answer and guarantees that that answer is right. McCabe-Dansted, Pritchard, and Slinko state that their DodgsonQuick is simpler than our `GreedyScore`. However, this is precisely because DodgsonQuick is just tossing out a score, rather than working to ensure self-knowingness. If one changed their DodgsonQuick to explicitly handle self-knowingness, it would become almost identical to our algorithm. Indeed, whenever our algorithm is self-knowingly correct, its computed score would coincide exactly with their algorithm's computed score—if one

[3]To avoid any possibility of confusion, we should mention that there seems to be a slight inching toward using the term "average-case tractability" to apply to frequency (or probability weight) of correctness claims (see also Impagliazzo's [Imp95] framework called heuristic polynomial time), even though such claims typically do not imply that any natural or transformed sort of averaging yields polynomial growth [Tre02, EHRS07]. [PR07] is an example of such a usage in a voting context and [BT06] is an example in a context far removed from voting. It is admittedly a matter of taste and of what is most natural in English, but we feel that to avoid the confusion of overloaded and nonintuitive terminology, it would be best to use the words "average-case" only when referring to true averaging or to Levin's averaging-related approach. When speaking of frequency (or probability weight) of correctness, those terms themselves, or speaking of the "usual-case complexity" ([CS06] use the word "usual" in their title, for example), or using the nomenclature of heuristic polynomial time [Imp95] are best. ("Typical-case complexity" would be a very attractive phrasing, but a prominent average-case researcher has already suggested using those words synonymously with Levin's "average-case complexity.") [EHRS07] provides additional discussion of these issues.

modified their algorithm to correct for the "third major difference" mentioned below.

The second major difference regards range of application. Our analysis leaves both the number of candidates and the number of voters as variables. By doing so, we produce a general bound flexible enough to ensure that the correctness (indeed, the self-knowing correctness) probability, under uniform-like inputs, goes asymptotically to one as long as the number of voters is at least some more-than-quadratic polynomial in the number of candidates. In sharp contrast, their claims cover just the subcase of this in which the number of candidates is held *constant*, while the number of voters increases.[4]

A third and easy-to-miss major difference is that our Dodgson score is indeed Dodgson's score notion: The number of swaps needed to make the candidate a strictly beat each other candidate in a head-on-head comparison (Dodgson's crucial words relating to this are "absolute majority" [Dod76]), i.e., become a Condorcet winner in the standard sense of that term. In contrast, [MPS06], without mentioning that it is differing from the settled notion of Dodgson score, uses as its definition of that concept a quite different score: The number of swaps needed to make the candidate *tie or* beat each other candidate in a head-

[4]We note in passing that the fixed-number-of-candidates case is a somewhat unusual choice of case to focus on regarding approximation, since for this case it has been long known that there is an exact polynomial-time algorithm, namely the Lenstra-method-based approach of Bartholdi, Tovey, and Trick [BTT89]. In addition to that case, namely the standard (i.e., nonsuccinct) case, McCabe-Dansted, Pritchard, and Slinko [MPS06] also look—again just for a fixed number, call it m, of candidates—at the issue of *succinct inputs*, a question that at first blush might seem to be untroubled by the Bartholdi, Tovey, and Trick [BTT89] algorithm mentioned above. In the succinct input case, inputs are assumed to be given as a list of counts of each vote type (there naturally are $m!$ possible vote types). Note that in the succinct model, the input size is $\Theta(\log n)$, where n is the number of voters. For this fixed-number-of-candidates, succinct-input case, McCabe-Dansted, Pritchard, and Slinko give an exact algorithm (which does in effect incorporate self-knowingness and the work it takes to achieve that) that runs in—under their proof's understanding—*worst-case time exponential* in the input size and runs in *expected time* (under uniform-like input distribution) *polynomial*—even linear in the right machine model—in the input size which, recall, is $\Theta(\log n)$. However, we mention that Faliszewski, Hemaspaandra, and Hemaspaandra ([FHH06a, p. 645], see also [FHH06b]) have already noted that the Bartholdi, Tovey, and Trick [BTT89] approach gives an exact, *worst-case polynomial time* in the $\Theta(\log n)$ input size algorithm for the succinct case of fixed-number-of-candidates Dodgson score.

Nonetheless, one may still find value in the McCabe-Dansted, Pritchard, and Slinko succinct-case, fixed-number-of-candidates algorithm. In particular, it has—in the right machine model, as degrees of polynomials depend on one's machine model—as they state a *linear* (in the $\Theta(\log n)$ input size) *expected* running time. In contrast, the Bartholdi, Tovey, and Trick [BTT89] algorithm, shifted to the succinct case as per the Faliszewski, Hemaspaandra, and Hemaspaandra comment, would seem to have a (worst-case, though quite possibly its expected time would not be better) running time that is at the very least quadratic in the $\Theta(\log n)$ input size. (The reason it would seem to be at least quadratic is a bit technical: The underlying attack uses Lenstra's brilliant algorithm for integer programming with a fixed number of variables [Len83], and even solving a single integer program instance using Lenstra's method (see [Dow03, Nie02]) involves a linear number of operations on linear-sized integers, where both of those "linear"s are relative to its input size, which in the succinct case here would be $\Theta(\log n)$.) Finally, we note that if one reconsiders the McCabe-Dansted, Pritchard, and Slinko succinct-case, fixed-number-of-candidates algorithm in light of the Faliszewski, Hemaspaandra, and Hemaspaandra comment about Bartholdi, Tovey, and Trick's Lenstra-based approach adapted to the succinct case, then one can correctly assert (for the thus adapted algorithm) *linear expected time* (relative to a uniform-like distribution) and *polynomial worst-case time*, all relative to the input size of $\Theta(\log n)$. Again, we stress that this entire footnote refers just to the very restricted case of fixing the number of candidates.

on-head comparison. For clarity, let us refer to this notion as variantDscore. (To see that this is what they define, one must carefully note their nonstandard definition of Condorcet winner.) The difference between Dodgson score and variantDscore is not just an "off by one" in the score. In fact, intervening candidates may have to be crossed to best close a shifted gap, and so variantDscore can vary from being the same as the Dodgson score in some cases to being nontrivially smaller in other cases. On the other hand, due to the fact that both their and our algorithms focus on cases where purely adjacent swaps, at most one per voter, can triumph, this difference has just a relatively mild reflection in the formulas and algorithms of the two papers. For example, it is this difference that results in their paper having expressions such as $\lceil z/2 \rceil$ in contrast with, in our paper, expressions such as $\lfloor z/2 \rfloor + 1$.[5]

Like approximation algorithms, our algorithms are time efficient even in the worst case. But approximation algorithms typically have worst-case guarantees on how far the answers they provide deviate from the optimal answers. We by contrast are only interested in how frequently the algorithms are correct. Even when we study optimization problems, as with `GreedyScore`, we make no guarantees on how close actual Dodgson scores are to the corresponding answers that `GreedyScore` provides in cases when the confidence is "maybe."

Additionally, the key feature that separates our work from each of the above-mentioned related frameworks is the "self-knowing" aspect of our work. The closest related concepts to this aspect of our analysis are probably those involving proofs to be verified, such as NP certificates and the proofs in interactive proof systems. Although our algorithms do

[5]The fact that [MPS06] is about variantDscore rather than Dodgson score causes two potential problems. The first, which is bad news for [MPS06], is that their succinct-case algorithm (footnote 4 above discusses the succinct case) potentially is invalid, since it in rare cases falls back to the Bartholdi, Tovey, and Trick algorithm and assumes that that algorithm has the same notion of Dodgson score as it does. The second, which is good news for [MPS06], is that if what they call Dodgson score (variantDscore) and what Bartholdi, Tovey, and Trick call Dodgson score differ, then our comments in footnote 4 above how their fixed-candidate work is related to the Bartholdi, Tovey, and Trick/Lenstra approach potentially become off-the-mark. Both these problems/inconsistencies are smoothed over by the following easy observations that we now make: (a) the Bartholdi, Tovey, and Trick algorithm can be modified to work with either Dodgson score or variantDscore (one sets which case the algorithm does simply by appropriately setting the values d_k of [BTT89, p. 162]), and (b) the fact that the Bartholdi, Tovey, and Trick/Lenstra approach, as noted by Faliszewski, Hemaspaandra, and Hemaspaandra, gives for the fixed-number-of-candidates case an algorithm running in time polynomial in the $\Theta(\log n)$ input size easily applies (with an immediately clear modification) to both the Dodgson score case and the variantDscore case. Finally, to be completely fair, we mention that though we feel that all three papers [Dod76, BTT89, Tid87] that McCabe-Dansted, Pritchard, and Slinko [MPS06] cite regarding the definition of Dodgson score in fact do define Dodgson score rather than variantDscore, in each case this depends on whether one's reading of such terms as "absolute majority," "beats," "greater than," and "defeats" allows ties to fall within the scope of the term; we feel it does not. Although for the case of Dodgson's paper ("absolute majority") and Tideman's ("the number is... greater than"), two of the three papers they cite, the case is pretty airtight that those papers are defining Dodgson score (and not variantDscore), for Bartholdi, Tovey, and Trick ("defeats" and "to become a Condorcet winner"—though [BTT89] at one point does write, though we feel the "unique" is just to emphasize that when they exist, Condorcet winners are unique—the strange phrase "a unique Condorcet winner") there is perhaps very slightly more wiggle room. In any case, our observations (a) and (b) smooth over the two potential problems mentioned at the start of this footnote.

We refer the reader to [MPS06] and [Tid87] itself for more on Tideman's early, insightful paper.

not provide actual certificates, one could reasonably require a transcript of the execution of either of our algorithms. With such a transcript, it would be easy to verify in deterministic polynomial time that the algorithm, in cases where the confidence is "definitely," presented a valid proof. Put somewhat differently, our algorithms can easily be modified to give, in each "definitely" case, a specific set of swaps that achieve a Dodgson score that is clearly optimal. By contrast, in interactive proof systems the methods for verifying the proofs involve a probabilistic, interactive process. We do not consider interactive processes in this paper. In Section 3 we discuss in more detail the differences between heuristics that find NP certificates and self-knowingly correct algorithms, but the most obvious difference is that, unless coNP = NP, the problem of verifying whether a given candidate is a Dodgson winner for a given election is not even in NP.

2 Dodgson Elections

As mentioned in the introduction, in Dodgson's voting system each voter strictly ranks the candidates in order of preference. Formally speaking, for $m, n \in \mathbb{N}^+$—throughout this paper we by definition do not admit as valid elections with zero candidates or zero voters—a *Dodgson election* is an ordered pair (C, V) where C is a set $\{c_1, \ldots, c_m\}$ of candidates (as noted earlier, we without loss of generality view them as being named by 1, 2, $\ldots$, m) and V is a tuple $(v_1, \ldots, v_n)$ of *votes* and a *Dodgson triple*, denoted (C, V, c), is a Dodgson election (C, V) together with a candidate $c \in C$. Each vote is one of the $m!$ total orderings over the candidates, i.e., it is a complete, transitive, and antireflexive relation over the set of candidates. We will sometimes denote a vote by listing the candidates in increasing order, e.g., (x, y, z) is a vote over the candidate set $\{x, y, z\}$ in which y is preferred to x and z is preferred to (x and) y. A candidate is never preferred to him- or herself. For vote v and candidates $c, d \in C$, "$c <_v d$" means "in vote v, d is preferred to c" and "$c \prec_v d$" means "$c <_v d$ and there is no e such that $c <_v e <_v d$." Each Dodgson election gives rise to $\binom{m}{2}$ *pairwise races*, each of which is created by choosing two distinct candidates $c, d \in C$ and restricting each vote v to the two chosen candidates, that is, to either (c, d) or (d, c). The winner of the pairwise race is the one that a strict majority of voters prefer. Due to ties, a winner may not always exist in pairwise races.

A *Condorcet winner* is any candidate c that, against each remaining candidate, is preferred by a strict majority of voters. For a given election (i.e., for a given sequence of votes), it is possible that no Condorcet winner exists. However, when one does exist, it is unique.

Dodgson's scores are in a sense an edit-distance notion based on the basic operation of swapping adjacent candidates in a voter's preference order. In particular, for any vote v and any $c, d \in C$, if $c \prec_v d$, let $\mathit{Swap}_{c,d}(v)$ denote the vote v', where v' is the same total ordering of C as v except that $d <_{v'} c$ (note that this implies $d \prec_{v'} c$). If $c \not\prec_v d$ then $\mathit{Swap}_{c,d}(v)$ is undefined. In effect, a swap causes c and d to "switch places," but only if c and d are adjacent. The *Dodgson score* of a Dodgson triple (C, V, c) is the minimum number of swaps that, applied sequentially to the votes in V, make V a sequence of votes in which c is the Condorcet winner. A *Dodgson winner* is a candidate that has the smallest

Dodgson score. This is the election system developed in the year 1876 by Dodgson (Lewis Carroll) [Dod76], and as noted earlier it gives victory to the candidate(s) who are “closest” to being Condorcet winners. Note that if no candidate is a Condorcet winner, then two or more candidates may tie, in which case all tying candidates are Dodgson winners.

Several examples show how Dodgson elections work, and hint at why they are hard, in general, to solve: Consider an election having four candidates $\{a, b, c, d\}$ and one hundred votes in which sixty are (a, b, c, d) and forty are (c, d, a, b). Since d is already (i.e., before any exchanges take place) a Condorcet winner, d's Dodgson score is 0. Thus d is the Dodgson winner.

Now suppose in another election having the same candidates and the same number of voters as in the previous example that twenty voters each vote (a, b, c, d), (b, c, d, a), (c, d, a, b), (b, a, d, c), and (d, a, b, c), respectively. In this case, there is no Condorcet winner, so we calculate the Dodgson score of each candidate. Consider candidate a. Candidate a beats d by twenty votes—of course, changes in the votes of as few as eleven voters can overcome such a shortfall. Candidate a loses to c by twenty votes. And candidate a loses to b by twenty votes. What is the fewest number of swaps needed to make a a Condorcet winner? It might be tempting to make, for each of eleven votes of the form (a, b, c, d), the following transformation: $(a, b, c, d) \rightarrow (d, b, c, a)$. But this transformation is not an allowed swap because only elements that are adjacent in the ordering imposed by the vote may be swapped at unit cost. We can, however, make eleven of the following series of two swaps each: $(a, b, c, d) \rightarrow (b, a, c, d) \rightarrow (b, c, a, d)$. This can clearly be seen to be an optimal number of swaps in light of a's initial vote shortfalls—and note that every swap improves a's standing against either b or c in a way that directly reduces a still-existing shortfall. So the Dodgson score of (C, V, a) is twenty-two.

What makes it hard to calculate Dodgson scores is what makes many combinatorial optimization problems hard: There is no apparent, simple way to locally determine whether a swap will lead to an optimally short sequence that makes the candidate of interest the Condorcet winner. For instance, if in calculating the Dodgson score of a we had come across votes of the form (b, a, d, c) first, we might have made the following series of swaps, $(b, a, d, c) \rightarrow (b, d, a, c) \rightarrow (b, d, c, a)$, which contain a swap between a and d. But this series of swaps is not optimal because a already beats d, and because, as we saw, there is already a series of twenty-two swaps available, where each swap helps a against some adversary that a has not yet beaten, that makes a a Condorcet winner. (However, there are instances of Dodgson elections in which the only way for a candidate to become a Condorcet winner is for it to swap with adversaries that the candidate is already beating, so one cannot simply ignore this possibility.) Assuming that we did not at first see the votes that constitute this optimal sequence and instead hastily made swaps that did not affect a's current standing against b or c, we could have, as soon as we had come across votes of the form (a, b, c, d), backtracked from the hastily made swaps that led toward a nonoptimal solution and, eventually, have correctly calculated the Dodgson score. But as the size of elections increases, the amount of backtracking that a naive strategy might need to make in order to correct for any nonoptimal swaps combinatorially explodes.

Of course, computational complexity theory can give evidence of hardness that is probably more satisfying than are mere examples. However, before turning to the computational complexity of Dodgson-election-related problems, a couple of preliminary definitions are in order. The class NP is precisely the set of all languages solvable in nondeterministic polynomial time. Θ_2^p can be equivalently defined either as the class of languages solvable in deterministic polynomial time with $\mathcal{O}(\log n)$ queries to an NP language or as the class of languages solvable in deterministic polynomial time via parallel access to NP. Θ_2^p was first studied by Papadimitriou and Zachos [PZ83], received its current name from Wagner [Wag90], and has proven important in many contexts. In particular, it seems central in understanding the complexity of election systems [HHR97,HH00,SV00,SV01,RSV03, HSV05]. All NP languages are in Θ_2^p. It remains open whether Θ_2^p contains languages that are not in NP; it does exactly if NP $\neq$ coNP.

Bartholdi, Tovey, and Trick [BTT89] define the following decision problems, i.e., mappings from Σ^* to {"yes", "no"}, associated with Dodgson elections. We take the problem wordings from [HHR97].

Decision Problem: `DodgsonScore`
Instance: A Dodgson triple (C, V, c); a positive integer k.
Question Is $Score(C, V, c)$, the Dodgson score of candidate c in the election specified by (C, V), less than or equal to k?

Decision Problem: `DodgsonWinner`
Input: A Dodgson triple (C, V, c).
Question: Is c a winner of the election? That is, does c tie-or-defeat all other candidates in the election?

Bartholdi, Tovey, and Trick show that the problem of checking whether a certain number of changes suffices to make a given candidate the Condorcet winner is NP-complete and that the problem of determining whether a given candidate is a Dodgson winner is NP-hard [BTT89]. Hemaspaandra, Hemaspaandra, and Rothe show [HHR97] that this latter problem, as well as the related problem of checking whether a given candidate is at least as close as another given candidate to being a Dodgson winner, is complete for Θ_2^p. Hemaspaandra, Hemaspaandra, and Rothe's results show that determining a Dodgson winner is not even in NP unless the polynomial hierarchy collapses. This line of work has significance because the hundred-year-old problem of deciding if a given candidate is a Dodgson winner was much more naturally conceived than the problems that were previously known to be complete for Θ_2^p (see [Wag87]).

3 The `GreedyScore` and `GreedyWinner` Algorithms

In this section, we study the polynomial-time greedy algorithms `GreedyScore` and `GreedyWinner`, which are given, respectively, as Figure 1 (page 13) and Figure 2 (page 14).

We show that both algorithms are self-knowingly correct in the sense of the following definition.

Definition 3.1. *For sets S and T and function $f : S \to T$, an algorithm $\mathcal{A} : S \to T \times \{$"definitely", "maybe"$\}$ is* self-knowingly correct *for f if, for all $s \in S$ and $t \in T$, whenever $\mathcal{A}$ on input s outputs (t, "definitely") it holds that $f(s) = t$.*

The reader may wonder whether "self-knowing correctness" is so easily added to heuristic schemes as to be uninteresting to study. For example, if one has a heuristic for finding certificates for an NP problem with respect to some fixed certificate scheme (in the standard sense of NP certificate schemes)—e.g., for trying to find a satisfying assignment to an input (unquantified) propositional boolean formula—then one can use the P-time checker associated with the problem to "filter" the answers one finds, and can put the label "definitely" on only those outputs that are indeed certificates. However, the problem studied in this paper does not seem amenable to such after-the-fact addition of self-knowingness, as in this paper we are dealing with heuristics that are seeking objects that are computationally much more complex than mere certificates related to NP problems. In particular, a polynomial-time function-computing machine seeking to compute an input's Dodgson score seems to require about logarithmically many adaptive calls to SAT.[6]

We call `GreedyScore` "greedy" because, as it sweeps through the votes, each swap it (virtually) does immediately improves the standing of the input candidate against some adversary that the input candidate is at that point losing to. The algorithm nonetheless is very simple. It limits itself to at most one swap per vote. Yet, its simplicity notwithstanding, we will eventually prove that this (self-knowingly correct) algorithm is very frequently correct.

Our results in this section, since they are just stated as simply polynomial-time results (see Theorem 3.2 below), are not heavily dependent on the encoding scheme used. However, we will for specificity give a specific scheme that can be used. Note that the scheme we use will encode the inputs as binary strings by a scheme that is easy to compute and invert and encodes each vote as an $\mathcal{O}(\|C\| \log \|C\|)$-bit substring and each Dodgson triple as an $\mathcal{O}(\|V\| \cdot \|C\| \cdot \log \|C\|)$-bit string, where (C, V, c) is the input to the encoding scheme. For a Dodgson triple (C, V, c), our encoding scheme is as follows.

- The first bits of the encoding string contain $\|C\|$, encoded as a binary string of length $\lceil \log(\|C\| + 1) \rceil$,[7] preceded by the substring $1^{\lceil \log(\|C\|+1) \rceil}0$.

[6]We say "seems to," but we note that one can make a more rigorous claim here. As mentioned in Section 2, among the problems that Hemaspaandra, Hemaspaandra, and Rothe [HHR97] prove complete for the language class Θ_2^p is `DodgsonWinner`. If one could, for example, compute Dodgson scores via a polynomial-time function-computing machine that made a (globally) constant-bounded number of queries to SAT, then this would prove that `DodgsonWinner` is in the boolean hierarchy [CGH$^+$88], and thus that Θ_2^p equals the boolean hierarchy, which in turn would imply the collapse of the polynomial hierarchy [Kad88]. That is, this function problem is so closely connected to a Θ_2^p-complete language problem that if one can save queries in the former, then one immediately has consequences for the complexity of the latter.

[7]All logarithms in this paper are base 2. We use $\lceil \log(\|C\| + 1) \rceil$-bit strings rather than $\lceil \log(\|C\|) \rceil$-bit

- Next in the encoding string are bits specifying the chosen candidate c, encoded as a binary string of length $\lceil \log(\|C\| + 1) \rceil$.

- The rest of the bits of the encoding string give the votes, where each vote is encoded as a binary substring of length $\|C\| \cdot \lceil \log(\|C\| + 1) \rceil$.

Recall that the `GreedyScore` and `GreedyWinner` algorithms are defined in Figures 1 and 2. We now describe what our algorithms do. Briefly put, on input (C, V, c), `GreedyScore` computes for each candidate $d \in C - \{c\}$ the number of votes by which d beats c in the pairwise contest between them, and the number of votes where $c \prec_v d$. These numbers are stored in *Deficit*$[d]$ and *Swaps*$[d]$, respectively. For example, if 17 voters prefer d to c and 3 voters prefer c to d, then *Deficit*$[d]$ will be set to 14. Note that in that case a shift of 8 fan-of-d voters would change the outcome to a victory for c in this pairwise contest. `GreedyScore` then takes as the Dodgson score

$$\|\{d \in C - \{c\} \mid \mathit{Deficit}[d] \geq 2 \times \mathit{Swaps}[d]\}\| + \sum_{d \in C - \{c\} : \mathit{Deficit}[d] \geq 0} (\lfloor \mathit{Deficit}[d]/2 \rfloor + 1).$$

`GreedyScore` outputs this number, paired with "definitely" if $(\forall d \in C - \{c\})[\mathit{Deficit}[d] < 2 \times \mathit{Swaps}[d]]$, or with "maybe," otherwise.[8]

Turning to the `GreedyWinner` algorithm, it does the above for all candidates, and if while doing so `GreedyScore` is never stumped, then `GreedyWinner` uses in the obvious way the information it has obtained, and (correctly) states whether c is a Dodgson winner of the input election.

Theorem 3.2. *1. `GreedyScore` is self-knowingly correct for Score.*

2. `GreedyWinner` is self-knowingly correct for `DodgsonWinner`.

3. `GreedyScore` and `GreedyWinner` both run in polynomial time.

Proof. In item 1's statement, as set in Section 2 in the statement of the `DodgsonScore` problem, *Score* denotes the Dodgson score. Now, suppose that `GreedyScore`, on input (C, V, c), returns "definitely" as the second component of its output. Then, as inspecting lines 29–33 of the algorithm makes clear, every candidate $d \in C - \{c\}$ for which *Deficit*$[d] \geq 0$ must also have *Deficit*$[d] < 2 \times$ *Swaps*$[d]$. In this case, note that the algorithm sets its *score*

strings as we wish to have the size of the coding scale at least linearly with the number of voters—even in the pathological $\|C\| = 1$ case, in which each vote carries no information other than about the number of voters.

[8]In the conference version of this paper [HH06] the algorithm—like the above modified algorithm—had the property that whenever the confidence was "definitely," the score that was output was $\sum_{d \in C - \{c\} : \mathit{Deficit}[d] \geq 0} (\lfloor \mathit{Deficit}[d]/2 \rfloor + 1)$. However, for the conference version's algorithm, whenever the confidence was "maybe" the algorithm output a value that was a strict lower bound on the true score—and so was certainly wrong. In contrast, the above algorithm in the "maybe" case outputs a half-hearted stab at the score—one that is sometimes too high, sometimes right, and sometimes too low. In terms of our theorems, the behavior on "maybe" cases doesn't matter, as no promises are made about such cases.

```
GreedyScore$(C, V, c)$
▷ $C$ is the set of candidates.
▷ $V$ is the list of votes.
▷ $c \in C$ is the candidate whose score the algorithm tries to compute.
for $d \in C - \{c\}$                  ▷ Initialize counter variables, which represent:
    do $Deficit[d] \leftarrow 0$        ▷     The number of votes by which $d$ is beating $c$.
       $Swaps[d] \leftarrow 0$          ▷     The number votes that may be "greedily"
                                       ▷        swapped against $d$.
▷ Count the votes. (Each vote $v$ is treated as an array where $v[1]$ is the least
▷        preferred candidate, $v[2]$ is the second least preferred candidate, and so
▷        on, and $v[length[v]]$ is the most preferred candidate.)
for each vote $v$ in $V$
    do $i \leftarrow 1$
       while $v[i] \neq c$              ▷ Subtract the "pairwise votes" for $c$.
           do $d \leftarrow v[i]$
              $Deficit[d] = Deficit[d] - 1$
              $i \leftarrow i + 1$
       if $i < length[v]$               ▷ Count the opportunities to "greedily swap."
         then $d \leftarrow v[i + 1]$
              $Swaps[d] \leftarrow Swaps[d] + 1$
       for $i \leftarrow i + 1$ to $length[v]$   ▷ Add the "pairwise votes" against $c$.
           do $d \leftarrow v[i]$
              $Deficit[d] = Deficit[d] + 1$
$confidence \leftarrow$ "definitely"
▷ Calculate the score. If there are enough greedily swappable votes to overcome
▷        all positive deficits, then the sum over one plus half of each positive deficit
▷        (rounded down) is certainly the Dodgson score of $c$.
$score \leftarrow 0$
for each $d \in C - \{c\}$
    do if $Deficit[d] \geq 0$
         then $score \leftarrow score + \lfloor Deficit[d]/2 \rfloor + 1$
              if $Deficit[d] \geq 2 \times Swaps[d]$
                then $confidence \leftarrow$ "maybe"
                     $score \leftarrow score + 1$
return $(score, confidence)$
```

Figure 1: The GreedyScore algorithm

```
GreedyWinner($C, V, c$)
 1  $\triangleright$ $C$ is the set of candidates.
 2  $\triangleright$ $V$ is the list of votes.
 3  $\triangleright$ $c \in C$. We wish to test whether $c$ is a Dodgson winner in election $(C, V)$.
 4  $(cscore, confidence) \leftarrow$ GreedyScore($C, V, c$)
 5  $winner \leftarrow$ "yes"
 6  for candidates $d \in C - \{c\}$
 7          do $(dscore, dcon) \leftarrow$ GreedyScore($C, V, d$)
 8              if $dscore < cscore$
 9                  then $winner \leftarrow$ "no"
10              if $dcon =$ "maybe"
11                  then $confidence \leftarrow$ "maybe"
12  return $(winner, confidence)$
```

Figure 2: The `GreedyWinner` algorithm

variable to $score = \sum_{d \in C - \{c\} : Deficit[d] \geq 0} (\lfloor Deficit[d]/2 \rfloor + 1)$. For this value of *score* to actually be the Dodgson score of c, we must show (a) that we can by performing *score* swaps turn (C, V) into an election in which c is the Condorcet winner and (b) that by performing fewer than *score* swaps we cannot make c a Condorcet winner. Both claims depend on the following observation: Let v be a vote having some candidate d such that $c \prec_v d$. Then swapping c and d would decrease by two the difference between the number of votes where c is preferred d and the number of votes where d is preferred c. Also, c's standing against any candidate other than d would not be affected by this swap.

So, regarding the number of swaps needed to make c beat some $d \in C - \{c\}$, let V_d be the set of votes in V in which $c \prec_c d$. If $Deficit[d]$ is, by line 29, nonnegative (if $Deficit[d]$ is negative then c already beats d and the number of swaps needed is trivially 0), and if $||V_d|| \geq \lfloor Deficit[d]/2 \rfloor + 1$, then we can make c beat d by choosing exactly $\lfloor Deficit[d]/2 \rfloor + 1$ votes in V_d and swapping the positions of c in d in these votes. But $Swaps[d]$ is precisely $||V_d||$ and is, by our $Deficit[d] < 2 \times Swaps[d]$ assumption, at least $\lfloor Deficit[d]/2 \rfloor + 1$. Thus, by summing over all such $d \in C - \{c\}$, *score* is enough swaps to make c a Condorcet winner, and we have satisfied (a).

To see (b), suppose that it is possible to make c a Condorcet winner with *fewer* than $\sum_{d \in C - \{c\} : Deficit[d] \geq 0} (\lfloor Deficit[d]/2 \rfloor + 1)$ swaps. Then for some $d \in C - \{c\}$ such that $Deficit[d] \geq 0$ it must hold that at most $\lfloor Deficit[d]/2 \rfloor$ of these swaps are applied to c's standing against d. But then, since as noted above one swap changes the deficit between c and d by exactly two, we do not have enough swaps to make c beat d. So we must conclude under our current supposition—namely, that `GreedyScore` returns "definitely" as the second component of its output—that *score* is the optimal number of swaps needed to make c a Condorcet winner.

For item 2, clearly `GreedyWinner` correctly checks whether c is a Dodgson winner if every call it makes to `GreedyScore` correctly calculates the Dodgson score. `GreedyWinner` then returns "definitely" exactly if each call it makes to `GreedyScore` returns "definitely." But, by item 1, `GreedyScore` is self-knowingly correct.

Item 3 follows from a straightforward analysis of the algorithm. The total number of line-executions in a run of `GreedyScore` is clearly $\mathcal{O}(\|V\| \cdot \|C\|)$, and for `GreedyWinner` is, including the line-executions within the subroutine calls, $\mathcal{O}(\|V\| \cdot \|C\|^2)$. So these are indeed polynomial-time algorithms.[9] □

Note that `GreedyWinner` could easily be modified into a new polynomial-time algorithm that, rather than checking whether a given candidate is the winner of the given Dodgson election, finds all Dodgson winners by taking as input a Dodgson election alone (rather than a Dodgson triple) and outputting a list of *all* the Dodgson winners in the election. This modified algorithm on any Dodgson election (C, V) would make exactly the same calls to `GreedyScore` that the current `GreedyWinner` (on input (C, V, c), where $c \in C$) algorithm makes, and the new algorithm would be accurate whenever every call it makes to `GreedyScore` returns "definitely" as its second argument. Thus, whenever the current `GreedyWinner` would return a "definitely" answer so would the new Dodgson-winner-finding algorithm (when their inputs are related in the same manner as described above). These comments explain why in the title (and abstract), we were correct in speaking of "*finding* Dodgson-Election Winners" (rather than merely recognizing them).

4 Analysis of the Correctness Frequency of the Two Heuristic Algorithms

In this section, we prove that, as long as the number of votes is much greater than the number of candidates, `GreedyWinner` is a frequently self-knowingly correct algorithm.

Throughout this section, regard $V = (v_1, \ldots, v_n)$ as a sequence of n independent observations of a random variable γ whose distribution is uniform over the set of all votes over a set $C = \{1, 2, \ldots, m\}$ of m candidates, where γ can take, with equal likelihood, any of the $m!$ distinct total orderings over C. (This distribution should be contrasted with such work as that of, e.g., [RM05], which in a quite different context creates dependencies between voters' preferences.)

The main intuition behind Theorem 4.1 is that, in any election having m candidates and n voters, and for any two candidates c and d, it holds that, in exactly half of the ways v a voter can vote, $c <_v d$, but for exactly $1/m$ of the ways, $c \prec_v d$. Thus, assuming that the

[9]For completeness, we mention that when one takes into account the size of the objects being manipulated (in particular, under the assumption—which in light of the encoding scheme we will use below is not unreasonable—that it takes $\mathcal{O}(\log \|C\|)$ time to look up a key in either *Deficit* or *Votes* and $\mathcal{O}(\log \|V\|)$ time to update the associated value, and each *Swap* operation takes $\mathcal{O}(\log \|C\|)$ time) the running time of the algorithm might be more fairly viewed as $\mathcal{O}(\|V\| \cdot \|C\| \cdot (\log \|C\| + \log \|V\|))$ (respectively, $\mathcal{O}(\|V\| \cdot \|C\|^2 \cdot (\log \|C\| + \log \|V\|)))$, though in any case it certainly is a polynomial-time algorithm.

votes are chosen independently of each other, when the number of voters is large compared to the number of candidates, with high likelihood the number of votes v for which $c <_v d$ will hover around $n/2$ and the number of votes for which $c \prec_v d$ will hover around n/m. This means that there will (most likely) be enough votes available for our greedy algorithms to succeed.

Theorem 4.1. *For each $m, n \in \mathbb{N}^+$, the following hold. Let $C = \{1, \ldots, m\}$.*

1. *Let V satisfy $\|V\| = n$. For each $c \in C$, if for all $d \in C - \{c\}$ it holds that $\|\{i \in \{1, \ldots, n\} \mid c <_{v_i} d\}\| \leq \frac{2mn+n}{4m}$ and $\|\{i \in \{1, \ldots, n\} \mid c \prec_{v_i} d\}\| \geq \frac{3n}{4m}$ then $\texttt{GreedyScore}(C, V, c) = (Score(C, V, c)$, "definitely").*

2. *For each $c, d \in C$ such that $c \neq d$, $\Pr((\|\{i \in \{1, \ldots, n\} \mid c <_{v_i} d\}\| > \frac{2mn+n}{4m}) \vee (\|\{i \in \{1, \ldots, n\} \mid c \prec_{v_i} d\}\| < \frac{3n}{4m})) < 2e^{\frac{-n}{8m^2}}$, where the probability is taken over drawing uniformly at random an m-candidate, n-voter Dodgson election $V = (v_1, \ldots, v_n)$ (i.e., all $(m!)^n$ Dodgson elections having m candidates and n voters have the same likelihood of being chosen).*

3. *For each $c \in C$, $\Pr(\texttt{GreedyScore}(C, V, c) \neq (Score(C, V, c)$, "definitely")$) < 2(m-1)e^{\frac{-n}{8m^2}}$, where the probability is taken over drawing uniformly at random an m-candidate, n-voter Dodgson election $V = (v_1, \ldots, v_n)$.*

4. *$\Pr((\exists c \in C)[\texttt{GreedyWinner}(C, V, c) \neq (\texttt{DodgsonWinner}(C, V, c)$, "definitely")$]) < 2(m^2 - m)e^{\frac{-n}{8m^2}}$, where the probability is taken over drawing uniformly at random an m-candidate, n-voter Dodgson election $V = (v_1, \ldots, v_n)$.*

Proof of Theorem 4.1. For item 1, $\frac{2mn+n}{4m} = \frac{n}{2} + \frac{n}{4m}$, so, if $\|\{i \in \{1, \ldots, n\} \mid c <_{v_i} d\}\| \leq \frac{2mn+n}{4m}$ then either c already beats d or if not then the defection of more than $\frac{n}{4m}$ votes from preferring-d-to-c to preferring-c-to-d would (if such votes exist) ensure that c beats d. If $\|\{i \in \{1, \ldots, n\} \mid c \prec_{v_i} d\}\| \geq \frac{3n}{4m}$ then (keeping in mind that we have globally excluded as invalid all cases where at least one of n or m equals zero) $\|\{i \in \{1, \ldots, n\} \mid c \prec_{v_i} d\}\| > \frac{n}{4m}$, and so GreedyScore will be able to make enough swaps (in fact, and this is critically important in light of the GreedyScore algorithm, there is a sequence of swaps such that any vote has at most one swap operation performed on it) so that c beats d. Item 2 follows from applying the union bound (which of course does not require independence) to Lemma 4.3, which is stated and proven below. Item 3 follows from item 1 and from applying item 2 and the union bound to

$$\Pr(\bigvee_{d \in C - \{c\}} ((\|\{i \in \{1, \ldots, n\} \mid c <_{v_i} d\}\| > \frac{2mn+n}{4m}) \vee (\|\{i \in \{1, \ldots, n\} \mid c \prec_{v_i} d\}\| < \frac{3n}{4m}))).$$

Item 4 follows from item 1 and from applying item 2 and the union bound to

$$\Pr(\bigvee_{c,d \in C \wedge c \neq d} ((\|\{i \in \{1, \ldots, n\} \mid c <_{v_i} d\}\| > \frac{2mn+n}{4m}) \vee (\|\{i \in \{1, \ldots, n\} \mid c \prec_{v_i} d\}\| < \frac{3n}{4m})))$$

(note that $\|\{(c, d) \mid c \in C \wedge d \in C \wedge c \neq d\}\| = m^2 - m$). ☐

We now turn to stating and proving Lemma 4.3, which is needed to support the proof of Theorem 4.1. Lemma 4.3 follows from the variant of Chernoff's Theorem [Che52] stated as Theorem 4.2.

Theorem 4.2 ([AS00])**.** *Let $X_1, \ldots, X_n$ be a sequence of mutually independent random variables. If there exists a $p \in [0,1] \subseteq \mathbb{R}$ such that, for each $i \in \{1, \ldots, n\}$,*

$$\begin{aligned}\Pr(X_i = 1-p) &= p,\\ \Pr(X_i = -p) &= 1-p,\end{aligned}$$

then for all $a \in \mathbb{R}$ where $a > 0$ it holds that $\Pr(\Sigma_{i=1}^{n} X_i > a) < e^{-2a^2/n}$.

Lemma 4.3. *1. $\Pr(\|\{i \in \{1, \ldots, n\} \mid c <_{v_i} d\}\| > \frac{2mn+n}{4m}) < e^{\frac{-n}{8m^2}}$.*

2. $\Pr(\|\{i \in \{1, \ldots, n\} \mid c \prec_{v_i} d\}\| < \frac{3n}{4m}) < e^{\frac{-n}{8m^2}}$.

Proof. 1. For each $i \in \{1, \ldots, n\}$, define X_i as

$$X_i = \begin{cases} 1/2 & \text{if } c <_{v_i} d,\\ -1/2 & \text{otherwise.}\end{cases}$$

Then $\|\{i \in \{1, \ldots, n\} \mid c <_{v_i} d\}\| > \frac{2mn+n}{4m}$ exactly if

$$\sum_{i=1}^{n} X_i > \frac{1}{2}\left(\frac{2mn+n}{4m}\right) - \frac{1}{2}\left(n - \frac{2mn+n}{4m}\right).$$

Since $\frac{1}{2}\left(\frac{2mn+n}{4m}\right) - \frac{1}{2}\left(n - \frac{2mn+n}{4m}\right) = \frac{n}{4m}$, setting $a = \frac{n}{4m}$ and $p = \frac{1}{2}$ in Theorem 4.2 yields the desired result.

2. For each $i \in \{1, \ldots, n\}$, define X_i as

$$X_i = \begin{cases} 1/m & \text{if } c \not\prec_{v_i} d,\\ 1/m - 1 & \text{otherwise.}\end{cases}$$

Then $\|\{i \in \{1, \ldots, n\} \mid c \prec_{v_i} d\}\| < \frac{3n}{4m}$ if and only if $\|\{i \in \{1, \ldots, n\} \mid c \not\prec_{v_i} d\}\| > n - \frac{3n}{4m}$ if and only if

$$\sum_{i=1}^{n} X_i > \frac{1}{m}\left(n - \frac{3n}{4m}\right) + \left(\frac{1}{m} - 1\right)\frac{3n}{4m}.$$

Since $\frac{1}{m}\left(n - \frac{3n}{4m}\right) + \left(\frac{1}{m} - 1\right)\frac{3n}{4m} = \frac{n}{4m}$, setting $a = \frac{n}{4m}$ and $p = 1 - \frac{1}{m}$ in Theorem 4.2 yields the desired result. □

Theorem 4.4 captures the exact statement of our main results that, in the introduction, we promised to establish.

Theorem 4.4. 1. *For each (election, candidate) pair it holds that if* ***GreedyWinner*** *outputs "definitely" as its second output component, then its first output component correctly answers the question, "Is the input candidate a Dodgson winner of the input election?"*

2. *For each $m, n \in \mathbb{N}^+$, the probability that a Dodgson election E selected uniformly at random from all Dodgson elections having m candidates and n votes (i.e., all $(m!)^n$ Dodgson elections having m candidates and n votes have the same likelihood of being selected) has the property that there exists at least one candidate c such that* ***GreedyWinner*** *on input (E, c) outputs "maybe" as its second output component is less than $2(m^2 - m)e^{\frac{-n}{8m^2}}$.*

Proof. Theorem 4.4.1 follows from Theorem 3.2.2. Theorem 4.4.2 follows from Theorem 4.1.4. □

5 Conclusion and Open Directions

The Dodgson voting system elegantly satisfies the Condorcet criterion. Although it is NP-hard (and so if P $\neq$ NP is computationally infeasible) to determine the winner of a Dodgson election or compute scores for Dodgson elections, we provided heuristics, `GreedyWinner` and `GreedyScore`, for computing winners and scores for Dodgson elections. We showed that these heuristics are computationally simple. We also showed that, for a randomly chosen election of a given size, if the number of voters is much greater than the number of candidates (although the number of voters may still be polynomial in the number of candidates), then we get with exceedingly high likelihood we get the right answer and we know that the answer is correct.

We consider the fact that one can prove extremely frequent, self-knowing success even for such simple greedy algorithms to be an *advantage*—it is good that one does not have to resort to involved algorithms to guarantee extremely frequent, self-knowing success. Nonetheless, it is natural to wonder to what degree these heuristics can be improved. What would be the effect of adding, for instance, limited backtracking or random nongreedy swaps to our heuristics? Regarding our analysis, in the distributions we consider, each vote is cast independently of every other. What about distributions in which there are dependencies between voters?

Though Definition 3.1 rigorously defines "self-knowingly correct," we have been using "frequent" more informally in uses such as "frequently self-knowingly correct"—the level of frequency is specified in each case simply by whatever frequency our results prove. It is natural to wonder whether one can state a general, abstract model of what it means to be frequently self-knowingly correct. That would be a large project (that we heartily commend as an open direction), and here we merely make a brief definitional suggestion for a very abstract case—in some sense simpler to formalize than Dodgson elections, as Dodgson elections have both a voter-set size and a candidate-set size as parameters, and have a domain that is not Σ^* but rather is the space of valid Dodgson triples—namely the case of function problems where the function is total and the simple parameter of input-length is considered the natural way to view and slice the problem regarding its asymptotics. Such a model is often appropriate in computer science (e.g., a trivial such problem—leaving tacit the issues of encoding integers as bit-strings—is $f(n) = 2n$, and harder such problems are $f(n)$ equals the number of primes less than or equal to n and $f(0^i) = \|\mathrm{SAT} \cap \Sigma^i\|$).

Definition 5.1. *Let A be a self-knowingly correct algorithm for $g : \Sigma^* \to T$.*

1. *We say that A is* frequently self-knowingly correct *for g if* $\lim_{n\to\infty} \frac{\|\{x\in\Sigma^n \mid A(x)\in T\times\{\text{“maybe”}\}\}\|}{\|\Sigma^n\|} = 0$.
2. *Let h be some polynomial-time computable mapping from $\mathbb{N}$ to the rationals. We say that A is h-*frequently self-knowingly correct *for g if* $\frac{\|\{x\in\Sigma^n \mid A(x)\in T\times\{\text{“maybe”}\}\}\|}{\|\Sigma^n\|} = O(h(n))$.

Since the probabilities that the above definition is tracking may be quite encoding dependent, the second part of the above definition allows us to set more severe demands regarding how often the heuristic (which, being self-knowingly correct, always has the right output when its second component is “definitely”) is allowed to remain uncommitted. One sharp contrast between this framework and Impagliazzo's [Imp95] notion of heuristic polynomial time is that his heuristic algorithms are not required to *ever* correctly assert their correctness. In his model, one will know that one is often correct, but on each specific run one in general will have no clue as to whether the proffered answer is correct.

Acknowledgments In an undergraduate project in one of our courses, G. Goldstein, D. Berlin, K. Osipov, and N. Rutar proposed a more complex greedy algorithm for Dodgson elections that possessed the self-knowing correctness property, and experimentally observed that their algorithm was often successful. The present paper was motivated by their exciting experimental insight, and seeks to prove rigorously that a greedy approach can be frequently successful on Dodgson elections. We also thank the anonymous conference and journal referees for their comments, which greatly improved the clarity and readability of this article.

References

[ACP95] G. Ausiello, P. Crescenzi, and M. Protasi. Approximate solution of NP optimization problems. *Theoretical Computer Science*, 150(1):1–55, 1995.

[AS00] N. Alon and J. Spencer. *The Probabilistic Method.* Wiley–Interscience, second edition, 2000.

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

[Bor84] J. de Borda. Mémoire sur les élections au scrutin. *Histoire de L'Académie Royale des Sciences Année 1781*, 1784.

[Bro01] D. Brown. A probabilistic analysis of a greedy algorithm arising from computational biology. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 206–207. ACM Press, 2001.

[BT06] A. Bogdanov and L. Trevisan. On worst-case to average-case reductions for NP problems. *SIAM Journal on Computing*, 36(4):1119–1159, 2006.

[BTT89] 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.

[CGH$^+$88] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy I: Structural properties. *SIAM Journal on Computing*, 17(6):1232–1252, 1988.

[Che52] H. Chernoff. A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23(4):493–509, 1952.

[CK76] L. Chang and J. Korsh. Canonical coin changing and greedy solutions. *Journal of the ACM*, 23(3):418–422, 1976.

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

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

[CS06] V. Conitzer and T. Sandholm. Nonexistence of voting rules that are usually hard to manipulate. In *Proceedings of the 21st National Conference on Artificial Intelligence*, pages 627–634. AAAI Press, July/August 2006.

[DF99] R. Downey and M. Fellows. *Parameterized Complexity.* Springer-Verlag, 1999.

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

[Dow03] R. Downey. Parameterized complexity for the skeptic. In *Proceedings of the 18th Annual IEEE Conference on Computational Complexity*, pages 147–168. IEEE Computer Society Press, July 2003.

[EHRS07] G. Erdélyi, L. Hemaspaandra, J. Rothe, and H. Spakowski. On approximating optimal weighted lobbying, and frequency of correctness versus average-case polynomial time. Technical Report TR-914, Department of Computer Science, University of Rochester, Rochester, NY, March 2007.

[FHH06a] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. The complexity of bribery in elections. In *Proceedings of the 21st National Conference on Artificial Intelligence*, pages 641–646. AAAI Press, July 2006.

[FHH06b] 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.

[FHHR07] P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Llull and Copeland voting broadly resist bribery and control. In *Proceedings of the 22nd National Conference on Artificial Intelligence*. AAAI Press, July 2007. To appear.

[GMS84] A. Goldberg and A. Marchetti-Spaccamela. On finding the exact solution of a zero-one knapsack problem. In *Proceedings of the 16th ACM Symposium on Theory of Computing*, pages 359–368. ACM Press, April 1984.

[Hem89] L. Hemachandra. The strong exponential hierarchy collapses. *Journal of Computer and System Sciences*, 39(3):299–322, 1989.

[HH00] E. Hemaspaandra and L. Hemaspaandra. Computational politics: Electoral systems. In *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science*, pages 64–83. Springer-Verlag *Lecture Notes in Computer Science #1893*, August/September 2000.

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

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

[HP01] G. Hägele and F. Pukelsheim. The electoral writings of Ramon Llull. *Studia Lulliana*, 41(97):3–38, 2001.

[HSV05] E. Hemaspaandra, H. Spakowski, and J. Vogel. The complexity of Kemeny elections. *Theoretical Computer Science*, 349(3):382–391, 2005.

[Imp95] R. Impagliazzo. A personal view of average-case complexity. In *Proceedings of the 10th Structure in Complexity Theory Conference*, pages 134–147. IEEE Computer Society Press, June 1995.

[Kad88] J. Kadin. The polynomial time hierarchy collapses if the boolean hierarchy collapses. *SIAM Journal on Computing*, 17(6):1263–1282, 1988. Erratum appears in the same journal, 20(2):404.

[KKL02] A. Kaporis, L. Kirousis, and E. Lalas. The probabilistic analysis of a greedy satisfiability algorithm. In *Proceedings of the 10th Annual European Symposium on Algorithms*, pages 574–585. Springer-Verlag *Lecture Notes in Computer Science #2461*, 2002.

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

[Lev86] L. Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–286, 1986.

[MPS06] J. McCabe-Dansted, G. Pritchard, and A. Slinko. Approximability of Dodgson's rule. In U. Endriss and J. Lang, editors, *Proceedings of the 1st International Workshop on Computational Social Choice*, pages 331–344. Universiteit van Amsterdam, December 2006. Available online at `staff.science.uva.nl/~ulle/COMSOC-2006/proceedings.html`.

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

[Nan82] E. Nanson. Methods of election. *Transactions and Proceedings of the Royal Society of Victoria*, 19:197–240, 1882.

[Nie02] R. Niedermeier. Invitation to fixed-parameter algorithms. Habilitation thesis, University of Tübingen, 2002.

[PR07] A. Procaccia and J. Rosenschein. Junta distributions and the average-case complexity of manipulating elections. *Journal of Artificial Intelligence Research*, 28:157–181, 2007.

[PT83] M. Protasi and M. Talamo. A new probabilistic model for the study of algorithmic properties of random graph problems. In *Proceedings of the 4th Conference on Fundamentals of Computation Theory*, pages 360–367. Springer-Verlag *Lecture Notes in Computer Science #158*, 1983.

[PZ83] C. Papadimitriou and S. Zachos. Two remarks on the power of counting. In *Proceedings of the 6th GI Conference on Theoretical Computer Science*, pages 269–276. Springer-Verlag *Lecture Notes in Computer Science #145*, 1983.

[RM05] G. Raffaelli and M. Marsili. Statistical mechanics model for the emergence of consensus. *Physical Review E*, 72(1):016114, 2005.

[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.

[SK83] D. Sankoff and J. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Computation.* Addison-Wesley, 1983.

[Sla97] P. Slavik. A tight analysis of the greedy algorithm for set cover. *Journal of Algorithms*, 25(2):237–254, 1997.

[ST04] D. Spielman and S. Teng. Smoothed analysis: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3):385–463, 2004.

[SV00] H. Spakowski and J. Vogel. Θ_2^p-completeness: A classical approach for new results. In *Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 348–360. Springer-Verlag *Lecture Notes in Computer Science #1974*, December 2000.

[SV01] H. Spakowski and J. Vogel. The complexity of Kemeny's voting system. In *Proceedings of the Workshop Argentino de Informática Teórica*, pages 157–168. Volume 30 of *Anales Jornadas Argentinas de Informática e Investigación Operativa*, SADIO, September 2001.

[Tid87] T. Tideman. Independence of clones as a criterion for voting rules. *Social Choice and Welfare*, 4(3):185–206, 1987.

[Tre02] L. Trevisan. Lecture notes on computational complexity. www.cs.berkeley.edu/~luca/notes/complexitynotes02.pdf (Lecture 12), 2002.

[Wag87] K. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theoretical Computer Science*, 51(1–2):53–80, 1987.

[Wag90] K. Wagner. Bounded query classes. *SIAM Journal on Computing*, 19(5):833–846, 1990.

[Wan97a] J. Wang. Average-case computational complexity theory. In L. Hemaspaandra and A. Selman, editors, *Complexity Theory Retrospective II*, pages 295–328. Springer-Verlag, 1997.

[Wan97b] J. Wang. Average-case intractable NP problems. In D. Du and K. Ko, editors, *Advances in Languages, Algorithms, and Complexity*, pages 313–378. Kluwer Academic Publishers, 1997.