Guarantees for the Success Frequency of an Algorithm for Finding Dodgson-Election Winners[1]

Christopher M. Homan and Lane A. Hemaspaandra

Abstract

Dodgson's election system elegantly satisfies the Condorcet criterion. However, determining the winner of a Dodgson election is known to be Θ_2^p-complete ([HHR97], see also [BTT89]), which 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

The *Condorcet paradox* [Con85], otherwise known as *the paradox of voting* or *the Condorcet effect*, says that rational (i.e., well-ordered) individual preferences can lead to irrational (i.e., cyclical) majority preferences.[2] It is a well-known and widely studied problem in the field of social choice theory [MU95]. A voting system is said to obey the *Condorcet criterion* [Con85] if whenever there is a Condorcet winner—a candidate who in each pairwise subcontest gets a strict majority of the votes—that candidate is selected by the voting system as the overall winner.

The mathematician Charles Dodgson (who wrote fiction under the now more famous name 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

[1]This paper appeared in technical report form as [HH06a] and a shorter version appeared as [HH06b]. Supported in part by NSF grant CCF-0426761, a Friedrich Wilhelm Bessel Research Award, the Alexander von Humboldt Foundation's TransCoop program, and an RIT Faculty Evaluation and Development grant.

[2]For instance, 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). Thus, each voter would have a cycle-free set of preferences, yet 2/3 of the voters would prefer b to a, another 2/3 would prefer a to c, and still another 2/3 would prefer c to b.

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 only in an unpublished pamphlet on the conduct of elections [Dod76] and may never have intended for it to be published. It was eventually discovered and disseminated by Black [Bla58] and is now regarded as a classic of social choice theory [MU95]. Dodgson's system was one of the first to satisfy the Condorcet criterion.[3]

Although Dodgson's system has many nice properties, it also poses a serious computational worry: 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 class of problems solvable with polynomial-time parallel access to an NP oracle [PZ83]. (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-theoretic 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 all but 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"} × {"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 have the following results.

Theorem 1.1. *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?"*

[3]The Condorcet criterion may at first glance seem easy to satisfy, but Nanson showed [Nan82] that many well-known voting systems—such as the rank-order system [Bor84] widely attributed to Borda (which Condorcet himself studied [Con85] in the same paper in which he introduced the Condorcet criterion), in which voters assign values to each candidate and the one receiving the largest (or smallest) aggregate value wins—fail to satisfy the Condorcet criterion.

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*[4]*) 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 winner of E, and moreover will "know" that it got those answers right. We call `GreedyWinner` a *frequently self-knowingly correct*[5] heuristic. (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.

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

[4]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.

[5]The full version of this paper [HH06a] contains a long discussion of how self-knowing correctness differs from other sorts of algorithmic analysis such as smoothed analysis and average-case complexity, but for space reasons we cannot include that here.

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. (Note: 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.

For any vote v and any $c, d \in C$, if $c \prec_v d$, let $Swap_{c,d}(v)$ 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 $Swap_{c,d}(v)$ 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.

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 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 greedy algorithms `GreedyScore` and `GreedyWinner`, stated as, respectively, Algorithm 1 (page 6) and Algorithm 2 (page 7), and we note that their running time is polynomial. 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. After all, 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.

[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.

Algorithm 1: `GreedyScore`(C, V, c) [n = number of voters; m = number of candidates]

```
for all d ∈ C−{c} do
  Deficit[d] ← 1 − ⌈n/2⌉
  Swaps[d] ← 0
end for
for all votes v[] in V do
  state ← "nocount"
  for all i ∈ (1, . . . , m) do
    if (state = "incrdef") ∨ (state = "swap") then
      Deficit[v[i]] ← Deficit[v[i]] + 1
      if state = "swap" then
        Swaps[v[i]] ← Swaps[v[i]] + 1
        state ← "incrdef"
      end if
    else if c = v[i] then
      state ← "swap"
    end if
  end for
end for
confidence ← "definitely"
score ← 0
for all d ∈ C−{c} do
  if Deficit[d] > 0 then
    score ← score + Deficit[d]
    if Deficit[d] > Swaps[d] then
      confidence ← "maybe"
      score ← score + 1
    end if
  end if
end for
return (score, confidence)
```

We now state the main result for this section, and a bit later we will briefly describe the algorithms in English.

Theorem 3.2. *1. `GreedyScore` (Algorithm 1) is self-knowingly correct for Score (recall that Score is defined in Section 2 in the statement of the `DodgsonScore` problem).*

2. `GreedyWinner` (Algorithm 2) is self-knowingly correct for `DodgsonWinner`*.*

3. `GreedyScore` and `GreedyWinner` both run in polynomial time.[7]

Note that Theorem 1.1.1 follows directly from Theorem 3.2.2. We will prove Theorem 1.1.2 in Section 4.

Theorem 3.2, since it just states polynomial time, is 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.

---

[7]The number of times lines of Algorithm 1 (respectively, Algorithm 2) are executed is clearly $\mathcal{O}(\|V\| \cdot \|C\|)$ (respectively, $\mathcal{O}(\|V\| \cdot \|C\|^2)$), and so these are indeed polynomial-time algorithms.

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.

- First comes $\|C\|$, encoded as a binary string of length $\lceil \log(\|C\|+1) \rceil$,[8] preceded by the substring $1^{\lceil \log(\|C\|+1) \rceil}0$.
- Next comes the chosen candidate c, encoded as a binary string of length $\lceil \log(\|C\|+1) \rceil$.
- Finally each vote is encoded as a binary substring of length $\|C\| \cdot \lceil \log(\|C\|+1) \rceil$.

Regarding the notation used in Algorithm 1: A vote is represented as an array $v[]$ of length m, where $m = \|C\|$. For each vote $v[]$, $v[1]$ is the least preferred candidate, $v[2]$ is the second least preferred candidate, and so on, and $v[m]$ is the most preferred candidate. $Swap_i(v)$ means that the ith and $(i+1)$st values in $v[]$ are swapped.

Algorithm 2: `GreedyWinner`(C, V, c)

(*cscore*, *confidence*) = `GreedyScore`(C, V, c)
winner ← "yes"
for all candidates $d \in C - \{c\}$ **do**
  (*dscore*, *dcon*) ← `GreedyScore`(C, V, d)
  if *dscore* < *cscore* **then**
    winner ← "no"
  end if
  if *dcon* = "maybe" **then**
    confidence ← "maybe"
  end if
end for
return (*winner*, *confidence*)

We now describe in English what our algorithms actually do (however, all references above and below to specific variables such as $v[]$, $Swap[]$, and $Deficit[]$, refer to their included pseudocode versions). Briefly put, `GreedyScore`, for each candidate d, $c \neq d \in C$, computes (in $Deficit[d]$) the number of votes (if any) that c needs to gain in order to have strictly more votes than d (in a pairwise contest between them), and computes (in $Swaps[d]$) the number of votes v in which d is immediately adjacent to and preferred to c ($c \prec_v d$). If the former number is strictly greater than zero and the latter number is at least as large as the former number, then it is the case that by adjacent swaps in exactly the former number of votes—when done in that number of votes chosen from among those votes v satisfying $c \prec_v d$—c can be with perfect efficiency (every swap pays off by reducing a positive shortfall) be changed to beating d. If the number values just stated are not the case, the `GreedyScore` algorithm declares that it is stumped by the current input. If it is stumped for no candidate d, $c \neq d \in C$, then it simply adds up the costs of defeating each other candidate, and is secure in the knowledge that this is optimal (see also the proof below).

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

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

the obvious way the information it has obtained, and (correctly) states whether c is a Dodgson winner of the input election.

Proof of Theorem 3.2. For item 1, suppose that `GreedyScore`, on input (C, V, c), returns "definitely" as the second component of its output. Then, at the point in time when the algorithm completes, it must hold that, for each $d \in C - \{c\}$, $Swaps[d] \geq Deficit[d]$. Note that for each $d \in C - \{c\}$, $Deficit[d]$ is initially set to $1 - \lceil \|V\|/2 \rceil$ and then is incremented once for every vote v in which d is preferred to c. As noted above, it follows that $Deficit[d]$, if it after that process is nonnegative,

will be set to the minimum number of votes v where the relationship $c <_v d$ needs to be reversed in order for c to beat d. Also as noted above, $Swap[d]$ will by the time all votes are visited be set to the number of votes v such that $c \prec_v d$. Thus, since $Swaps[d] \geq Deficit[d]$ it is possible (i.e., by swapping c and d in $Deficit[d]$ of the votes that $Swaps[d]$ counts) to turn (C, V) into an election in which c beats d by performing only $Deficit[d]$ swaps involving d (which clearly is the fewest swaps that can result in c beating d) when $Deficit[d] > 0$, and by performing zero swaps involving d when $Deficit[d] \leq 0$. From this, and because for each $d, e \in C - \{c\}$ such that $d \neq e \wedge Swaps[d] \geq Deficit[d] > 0 \wedge Swaps[e] \geq Deficit[e] > 0$ it holds that $\{v \mid v \text{ is vote in } C \text{ and } c \prec_v e\} \cap \{v \mid v \text{ is vote in } C \text{ and } c \prec_v d\} = \emptyset$, one can by making $Deficit[d] + Deficit[e]$ swaps turn (C, V) into an election in which c beats both d and e. Similarly, one can by making $\Sigma_{d \in C - \{c\} : Deficit[d] > 0} Deficit[d]$ swaps turn (C, V) into an election in which c beats every $d \in C - \{c\}$. Because one swap reverses the preference relationship between exactly one pair of candidates in exactly one vote, $\Sigma_{d \in C - \{c\} : Deficit[d] > 0} Deficit[d]$ is the Dodgson score of c, which is the first component of the output of `GreedyScore` whenever the second component is "definitely."

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 (see also footnote 7). □

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.

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* $\texttt{GreedyScore}(C, V, c) = (\mathit{Score}(C, V, c),$ *"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 $c \in C$,* $\Pr(\texttt{GreedyScore}(C, V, c) \neq (\mathit{Score}(C, V, c),$ *"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 $V = (v_1, \ldots, v_n)$.*

Note that Theorem 1.1.2 follows from Theorem 4.1.4.

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 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.

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.)

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 Algorithm 1, 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 following variant of Chernoff's Theorem [Che52].

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 $i \in \{1,\ldots,n\}$, $(\Pr(X_i = 1-p) = p$ and $\Pr(X_i = -p) = 1-p)$, 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. □

We now have proven Theorem 1.1.

Proof of Theorem 1.1. As mentioned in Section 3, Theorem 1.1.1 follows from Theorem 3.2.2. Theorem 1.1.2 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, and we showed that, over all elections of a given size where 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) in a randomly chosen election, these algorithms, with likelihood approaching one, get the right answer and know that they are correct.

We consider the fact that one can prove this 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 success. Nonetheless, it is also 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?

It is also 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.

Acknowledgments In an undergraduate project in one of our courses, G. Goldstein, D. Berlin, K. Osipov, and N. Rutar proposed a (far more complex) greedy algorithm for Dodgson elections and experimentally observed that it 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 thank the anonymous MFCS06 and COMSOC06 referees for helpful comments.

References

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

[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*, 6:157–165, 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*, 23:493–509, 1952.

[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.

[Dod76] C. Dodgson. A method of taking votes on more than two issues. Clarendon Press, Oxford, pamphlet, 1876.

[HH06a] C. Homan and L. Hemaspaandra. Guarantees for the success frequency of an algorithm for finding Dodgson-election winners. Technical Report TR-2005-881, University of Rochester Department of Computer Science, June 2006.

[HH06b] C. Homan and L. Hemaspaandra. Guarantees for the success frequency of an algorithm for finding Dodgson-election winners. In *Proceedings of the 31st International Conference on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science #4162, pages 528–539. Springer-Verlag, 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.

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

[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.

[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.

[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.

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

Christopher M. Homan
Department of Computer Science
Rochester Institute of Technology
Rochester, NY 14623 USA
URL: `http://www.cs.rit.edu/~cmh`

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