Quickly Determining Who Won an Election

Lisa Hellerstein
Department of Computer Science and Engineering, New York University, Tandon School of Engineering, New York, United States

Naifeng Liu
Department of Computer Science, CUNY Graduate Center, New York, United States
Department of Economics, University of Mannheim, Mannheim, Germany

Kevin Schewior
Department of Computer Science and Mathematics, University of Southern Denmark, Odense, Denmark

Abstract

This paper considers elections in which voters choose one candidate each, independently according to known probability distributions. A candidate receiving a strict majority (absolute or relative, depending on the version) wins. After the voters have made their choices, each vote can be inspected to determine which candidate received that vote. The time (or cost) to inspect each of the votes is known in advance. The task is to (possibly adaptively) determine the order in which to inspect the votes, so as to minimize the expected time to determine which candidate has won the election. We design polynomial-time constant-factor approximation algorithms for both the absolute-majority and the relative-majority version. Both algorithms are based on a two-phase approach. In the first phase, the algorithms reduce the number of relevant candidates to $O(1)$, and in the second phase they utilize techniques from the literature on stochastic function evaluation to handle the remaining candidates. In the case of absolute majority, we show that the same can be achieved with only two rounds of adaptivity.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases stochastic function evaluation, voting, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ITCS.2024.60

Funding *Lisa Hellerstein*: Partially supported by NSF Award IIS-1909335.
Naifeng Liu: Partially supported by the Teaching Fellowship at City University of New York.
Kevin Schewior: Partially supported by the Independent Research Fund Denmark, Natural Sciences, grant DFF-0135-00018B.

Acknowledgements The authors thank R. Teal Witter and Devorah Kletenik for helpful discussions.

1 Introduction

We introduce the following problem. Consider an election with voters $1, \ldots, n$ who each vote for a single candidate out of $1, \ldots, d$. Each voter i chooses each candidate j with known probability $p_{i,j} \in (0, 1)$, independently of all other voters. A candidate wins the election if they receive a strict majority of the votes which may be, depending on the model, an absolute or a relative majority. After the votes have been collected, it is possible to count the vote of any voter i, taking a known time $c_i \geq 0$ (which can also be thought of as cost). The present paper is concerned with the following question: In what order should votes be counted so as to minimize the expected time (or cost) until it is known who won the election? We allow adapting this order along the way.

The above problem is a stochastic function-evaluation problem (for a survey, see [20]): The votes can be regarded as random variables taking one of d values, and the function maps these variables to the index of the winning candidate or, if no such candidate exists, to 0.

When $d = 2$, results on such problems can be applied to our problem. Note that for $d = 2$, there is no difference between absolute and relative majority. If n is odd, the problem for $d = 2$ is somewhat simpler because function value 0 is impossible. The function can then be seen as a so-called k-of-n function, a Boolean function that is 1 if and only if at least k of the n variables have value 1. For these functions, a beautiful optimal strategy due to Salloum, Breuer, and (independently) Ben-Dov is known [19, 4]. In what follows, we refer to this as the SBB strategy. In our setting, it translates to the following: An interchange argument shows that, if it were known a-priori that candidate j is the winner, it would be optimal to inspect votes in increasing order of $c_i/p_{i,j}$ ratios. Since the two strategies, for $j \in \{1, 2\}$, have to count $\lfloor n/2 \rfloor + 1$ votes before they have verified the winner, there exists, by the pigeonhole principle, a vote that is contained in both corresponding prefixes. An unconditionally optimal strategy can safely inspect this vote and then recurse on the resulting subinstance. In the case that $d = 2$ and n is even, determining who is the winner of the election, or determining that there is no winner because of a tie, is equivalent to determining whether there are precisely $n/2$ votes for both candidates, and an optimal strategy can be derived in a similar way [1, 14].

For our general setting, where d can be greater than 2, we do not expect a similarly clean strategy. For previously studied functions of Boolean variables that are slight extensions of k-of-n functions, the stochastic evaluation problem is either known to be NP-hard or no polynomial-time algorithms[1] computing optimal strategies are known. A recent trend in stochastic function evaluation has been to design (polynomial-time) *approximation* algorithms instead, i.e., algorithms whose expected cost can be bounded with respect to the expected cost of an optimal strategy, preferably by a constant factor [12, 14, 16, 18, 17, 8].

The problems considered in this paper have similarities to the stochastic function-evaluation problem for Boolean linear threshold functions. Constant-factor approximation algorithms have been developed for that problem, using two different approaches. The first approach reduces the problem to Stochastic Submodular Cover through the construction of a utility function and solves the problem with an algorithm called Adaptive Dual Greedy [8]. We show that this approach yields an $O(d)$-approximation for our problems (but it is conceivable that a more nuanced analysis could yield a better approximation factor). The second approach generates a separate strategy for each of the possible values of the function, by solving multiple instances of a constrained maximization problem, and interweaves the resulting strategies together [12]. Applying this approach and current analysis to our problems would involve interweaving at least d separate strategies, leading to (at least) a linear dependence on d.

Instead, we develop a two-phase approach, where the first phase reduces the number of relevant candidates to $O(1)$ and the second phase handles the remaining candidates with some of the aforementioned techniques. As we show, this approach yields $O(1)$-approximation algorithms for both the absolute-majority and the relative-majority setting.

A strategy is considered particularly useful in practice if it is non-adaptive (e.g., [15, 16, 12]), i.e., it considers votes in a pre-specified order until the function is evaluated. For the absolute-majority version, we give an $O(1)$-approximation algorithm with only two rounds of adaptivity. In counting rounds, we use the "permutation" definition: In each round, votes

[1] For an algorithm for our problem to run in polynomial time, we require that, in every situation, it can find the next vote to count in time polynomial in the input size. Note that the corresponding decision tree may have exponential size. For an in-depth discussion of such issues, we refer to the survey by Ünlüyurt [20, Section 4.1].

are inspected in an order that is pre-specified for that round, until some stopping condition is reached. This definition is consistent with usage in a number of previous papers (e.g., [2, 11]).

For future work, we leave it open whether there are non-adaptive $O(1)$-approximate strategies and, if so, whether they can be computed in polynomial time.

1.1 Our Contribution

We start with discussing the arguably simpler absolute-majority case. Here, a candidate wins if they receive more than $n/2$ votes. Let us first formalize the termination condition, i.e., the condition that we have enough information to determine the outcome of the election. The condition is fulfilled if it is either known that (i) some particular candidate has won or that (ii) it is no longer possible for any candidate to win. Clearly, (i) is fulfilled if, for some candidate j, the number of votes candidate j has received is at least $\lfloor n/2 \rfloor + 1$ and (ii) is fulfilled if, for all candidates j, the number of votes received by candidates other than j has reached at least $\lceil n/2 \rceil$.

Such conditions can be turned into submodular functions which map any state b (representing the values of the votes counted so far) to a non-negative integer value such that reaching a state whose value is equal to a certain goal value is equivalent to satisfying the condition. For instance, as noted before, we can conclude that candidate j is not the absolute-majority winner if and only if the number of votes for candidates other than j has reached at least $\lceil n/2 \rceil$. We will later denote the number of votes for other candidates in state b, capped at $\lceil n/2 \rceil$, by $g_j(b)$. By previous techniques (cf. [8]), such conditions can be combined to obtain a submodular function g with the property that, once g has reached a certain value, the termination condition is reached. As mentioned before, Adaptive Dual Greedy can then be applied to the resulting submodular function, and the approximation bound on Adaptivity Dual Greedy can then be shown to yield an approximation factor of $O(d)$. For completeness, we detail this analysis in the full version of the paper.

A second approach is based on the previously mentioned approach for $d = 2$. Note that, using the SBB strategy, it is possible to evaluate at minimum expected cost whether any fixed candidate j wins (by what can be viewed as merging the other candidates into a single one). Clearly, the expected cost of this strategy does not exceed the expected cost of an optimal strategy, as an optimal strategy also has to evaluate whether candidate j wins (and more than that). Since concatenating the corresponding strategies (where repeated votes are skipped) determines who is the winner, the resulting strategy has an approximation factor of at most d. Unfortunately, such an approximation factor seems inherent to an approach which consists of separate strategies for all d candidates, even if a more sophisticated round-robin approach (e.g., [3, 18]) is used to merge them into a single strategy.

As a third approach, let us try something much more naive, namely inspecting votes in increasing order of cost. Not surprisingly, this alone does not yield a constant-factor approximation algorithm, as shown, e.g., by the following instance: Consider $d = 2$ with n odd, and the following three types of votes: (i) $(n-1)/2$ votes i with $p_{i,1} = 1 - \varepsilon$ and $c_i = \varepsilon$, (ii) $(n-1)/2$ votes i with $p_{i,1} = \varepsilon$ and $c_i = 1 - \varepsilon$, and (iii) a special vote $i^\star$ with $p_{i^\star,1} = 1 - \varepsilon$ and $c_{i^\star} = 1$. When one considers votes in increasing order of cost, for $\varepsilon \to 0$, with probability approaching 1 it will be necessary to inspect all n votes (at a total cost approaching $(n+1)/2$). However, one could instead first inspect the type (i) votes, and then the special vote, and with probability approaching 1 this would be sufficient (at a total cost approaching 1).

Interestingly, a combination of the two latter approaches yields a constant-factor approximation algorithm. The crucial observation is that inspecting votes in increasing order of cost

as long as more than two candidates can still win does not incur more cost than a constant factor times the cost that an optimal strategy incurs. To get an intuition for why this is true, consider the case in which all costs are either ε (for negligible ε) or 1. After all cost-ε votes have been inspected, resulting in state b, let m_2 be the second-smallest distance of a function $g_j(b)$ from its goal value. It is not difficult to see that an optimum strategy still has to inspect at least m_2 votes. On the other hand, assume that, after inspecting $2m_2$ votes, there are still more than two functions $g_j(b)$ that have not reached their goal values yet. Noting that, for any pair of such goal functions, inspecting any vote adds 1 to at least one of them readily yields a contradiction. Interestingly, using a more involved charging argument, we are able to extend such an argument to arbitrary cost.

Next, consider the resulting situation, in which there are only two candidates left that can still win, say, candidates 1 and 2. We can now use the SBB strategy to first evaluate whether candidate 1 wins and then, again, to evaluate whether candidate 2 wins. Similarly to the argument given before, it is easy to argue that the expected cost of this is no larger than twice the expected cost of an optimal strategy. The resulting approximation guarantee of the entire algorithm is therefore 4. By replacing each instance of the SBB strategy with a 2-approximation non-adaptive strategy during the process of evaluating if candidate 1 wins and then if candidate 2 wins, we obtain an algorithm with only three rounds of adaptivity at the cost of a larger approximation factor of 6. Additionally, we give another algorithm that further reduces the rounds of adaptivity to 2 with a slightly subtle analysis, but it increases the approximation factor to 10.

We now discuss the relative-majority case. Here, a candidate wins if they receive more votes than any other candidate. Again, the termination condition is fulfilled if it is either known that (i) some particular candidate has won or that (ii) it is no longer possible for any candidate to win. To formalize this, consider any state b and any two candidates j and k. Note that j is known to have received more votes than candidate k if the number of inspected votes for candidate k plus the number of uninspected votes is not larger than the number of inspected votes for candidate j. Again, we can translate this to a goal-value function $g_{j,k}(b)$. Then, (i) is fulfilled if, for some candidate j, the $d-1$ functions $g_{j,k}(b)$ (for $k \in \{1,\dots,d\} \setminus \{j\}$) have reached their goal values. For (ii), all votes have to be inspected.

Interestingly, a similar approach works: Again, we inspect votes in increasing order of costs until only two candidates can win. We can bound the total cost of this phase by 4 times the cost of an optimal strategy. In the second phase, assume candidates 1 and 2 can still win. We would like to evaluate whether $g_{1,2}(b)$ reaches its goal value. To do so, we use Adaptive Dual Greedy. We adapt the analysis of Deshpande et al. for Boolean linear threshold functions [8], to handle the fact each vote may contribute 0, 1, or 2 to $g_{1,2}(b)$. As a consequence, we show that the cost is at most 3 times the expected cost of an optimal strategy. After this algorithm has been applied, it is either known that candidate 1 is the winner, or candidate 2 is the only candidate that can still possibly win. We use the SBB strategy to handle the latter case. The resulting total approximation factor is 8.

1.2 Further Related Work

The stochastic function-evaluation problem we discussed is inspired by the seminal work of Charikar et al. [7], where each input value is originally unknown but can be revealed by paying an *information price* c_i, and the goal is to design a query strategy to compute the value of function f with minimum cost. However, in the work of Charikar et al., the distributions of input variables are unknown so the performance is measured using competitive analysis. In our *stochastic* setting, we can compute the expected cost of query strategies since the

distributions are known, which allows us to use the standard analysis in approximation algorithms. Recently, Blanc et al. [5] revisited the priced information setting of Charikar et al., and they focus on a similar model to the Stochastic Boolean Function Evaluation problem of Deshpande et al. [8], but allow an ϵ error in the evaluation result.

Most recent works related to stochastic discrete minimization problems concentrate on designing non-adaptive strategies [12, 13, 16] or strategies that require small rounds of adaptivity [2, 11]. Non-adaptive strategies have advantages in many practical applications. For example, they typically require less storage space and they are amenable to parallelization. In addition, the *adaptivity gap* is studied to measure the performance difference between an optimal adaptive algorithm and an optimal non-adaptive algorithm. So if the adaptivity gap is small, non-adaptive strategies are usually more desired. However, there are situations where it is hard to design a non-adaptive strategy with good performance, e.g., when the adaptivity gap is large. Surprisingly, sometimes allowing a few rounds of adaptivity can greatly reduce the expected cost of an optimal non-adaptive strategy. Hence, it is interesting to design strategies that permit small rounds of adaptivity to keep a subtle balance between performance, storage space, and parallelizability.

Interestingly, economists study similar problems in deciding the winner of an election while the goal is different [6, 10]. For example, in one of the settings of finding a first-best mechanism, they seek to maximize the *social welfare*, which is the expected utility gain of individuals if the value of f matches the input of any coordinate, minus the expected cost spent in learning the input values that are needed to determine the value of f. In other words, this approach generates a complicated rule for determining the value of election results. Our approach can be considered as fixing the decision rule so the utility gain is constant, and now the goal is equivalent to minimizing the cost of learning the value of f. Hence, our algorithms can be potentially helpful in mechanism-design problems.

1.3 Overview

In Section 2, we give some needed definitions. In Sections 3 and 4, we give constant-factor approximation algorithms for the absolute-majority and the relative-majority cases, respectively. We conclude and state open problems in Section 5.

2 Preliminaries

Let n be the number of voters, and d be the number of candidates. We number the voters from 1 to n, and the candidates from 1 to d. We sometimes use $[k]$ to denote the set $\{1, \dots, k\}$.

For $X = (X_1, \dots, X_n) \in \{1, \dots, d\}^n$ we define $N_j(X)$ to be the number of entries X_i of X that are equal to j. The *absolute-majority function* $f : \{1, \dots, d\}^n \to \{0, 1, \dots, d\}$ is defined as follows:

$$f(X) = \begin{cases} j & \text{if } N_j(X) \geq \lfloor \frac{n}{2} \rfloor + 1, \\ 0 & \text{otherwise.} \end{cases}$$

Here $f(X) = j$ means j is an absolute-majority winner, meaning j receives more than half the votes, and $f(X) = 0$ means there is no absolute-majority winner. The *relative-majority function* $f : \{1, \dots, d\}^n \to \{0, \dots, d\}$ is as follows:

$$f(X) = \begin{cases} j & \text{if } \forall k \in \{1, \dots, d\} \setminus \{j\},\ N_j(X) > N_k(X), \\ 0 & \text{otherwise.} \end{cases}$$

Similar to the previous function, the output of f is either the winner of the election, or 0 if there is no winner. Here the winner must receive more votes than any other candidate.

The above two functions are symmetric functions, meaning that the results of the function only depend on the *number* of 1's, ..., d's in the input. (This property is called anonymity in social-choice theory.) We assume that the vote of voter i is an independent random variable X_i, taking a value in the set $\{1, \ldots, d\}$. For each candidate j, we use $p_{i,j}$ to denote the probability that $X_i = j$. We assume that each $p_{i,j} > 0$.

In our vote-inspection problems, we are given as input the $p_{i,j}$ values for each voter i and candidate j. For each voter i, we are also given a value $c_i \geq 0$. The only way to discover the value of X_i, the vote of voter i, is to "inspect" vote i, which incurs cost c_i. (We will sometimes use the phrase "test variable X_i" instead of "inspect vote i".) In other words, c_i is the cost of inspecting the vote of voter i. The problem is to determine the optimal (adaptive) order in which to sequentially inspect the votes, so as to minimize the expected total cost of determining the winner of the election.

An assignment a to the n variables X_i is a vector $a \in \{1, \ldots, d\}^n$, where a_i is the value assigned to x_i. We use assignments to represent the values of the votes of the n voters. A partial assignment is a vector $b \in \{1, \ldots, d, *\}^n$, where $b_i = *$ indicates that the value of X_i has not been determined. In discussing vote inspection strategies, we use a partial assignment b to represent the current state of knowledge regarding the values of the votes inspected so far, with b_i equal to the value of voter i's vote if it has been inspected already, and $b_i = *$ otherwise. In the same way as for X, we use $N_j(b)$ to denote the number of entries i of b such that $b_i = j$. So $N_j(b)$ represents the number of inspected votes that are for candidate j.

3 Absolute Majority

We describe an adaptive 4-approximation algorithm for the absolute-majority version of our problem. The general idea is as follows. Throughout, we keep a partial assignment b that records the vote values known so far. For each candidate j, let $g_j(b)$ denote the number of known votes for candidates other than j, capped at $\lceil n/2 \rceil$. We can rule out j as the absolute-majority winner of the election iff $g_j(b) = \lceil n/2 \rceil$. The algorithm has two phases. In the first phase, we inspect the votes i in increasing order of their costs c_i, until there are at most two candidates j who could still get enough votes to win the election. Equivalently, Phase 1 ends when the third-smallest value of $g_j(b)$, for all j, is $\lceil n/2 \rceil$. At this point, if we have enough information to determine the outcome of the election, we are done. Otherwise, for each of the at most two remaining potential winners j, we need to determine whether the remaining uninspected votes include enough additional votes for candidate j to make j the absolute-majority winner. (If neither do, then the election has no absolute-majority winner.) We take one of these remaining potential winners, calculate the minimum number k of votes they would need to win, and use the SBB strategy for evaluating k-of-n functions to determine whether this candidate is the winner. If not, and if another candidate is still a potential winner, we again use the SBB strategy to check if that candidate is the winner. If not, we know there is no winner. Pseudocode for the algorithm is presented in Algorithm 1 (note that we are more verbose than necessary regarding the output, for readability).

We now prove an approximation bound on the algorithm.

▶ **Theorem 1.** *Algorithm 1 is an adaptive 4-approximation algorithm for evaluating the absolute-majority function.*

Proof. We analyze the two phases of Algorithm 1: Phase 1 ends when the third-smallest $g_j(b)$ equals $\lceil n/2 \rceil$; note that Algorithm 1 cannot terminate before that happens. Phase

Algorithm 1 A 4-apx. adaptive algorithm for evaluating the absolute-majority function

```
b ← {*}^n
while the value of f is not certain do
    if the 3rd-smallest g_j(b) < ⌈n/2⌉ then
        test the next untested variable in increasing order of c_i
        update b
    else
        α ← any j ∈ arg min_{j∈{1,...,d}} g_j(b)
        β ← any j ∈ arg min_{j∈{1,...,d}\{α}} g_j(b)
        evaluate if f = α using the SBB strategy
        update b
        if f = α then
            return f = α
        else
            if g_β(b) = ⌈n/2⌉ then
                return f = 0
            else
                evaluate if f = β using the SBB strategy
                if f = β then
                    return f = β
                else
                    return f = 0
return the value of f
```

2 ends when the algorithm terminates. The key to our proof is to show that, in Phase 1, Algorithm 1 spends at most 2 times the cost of an optimal strategy.

We first consider the following situation. Suppose we are in the process of executing Algorithm 1 and Phase 1 has not yet ended. Let b' be the current value of partial assignment b, and without loss of generality, assume $N_1(b') \geq \cdots \geq N_d(b')$. Let $m_1(b), \ldots, m_d(b)$ denote the distances of functions $g_1(b), \ldots, g_d(b)$ to $\lceil n/2 \rceil$, i.e.,

$$\forall j \in \{1, \ldots, d\}, \quad m_j(b) = \lceil n/2 \rceil - \min \left\{ \sum_{\ell \in \{1,\ldots,d\} \setminus \{j\}} N_\ell(b), \lceil n/2 \rceil \right\}.$$

Since $N_1(b') \geq \cdots \geq N_d(b')$, we have $m_1(b') \geq \cdots \geq m_d(b')$. Since Phase 1 has not yet ended, $m_3(b') > 0$. Let S' denote the set of variables tested so far by Algorithm 1.

We will show that the following two facts hold:

- Fact 1: During its execution, an optimal algorithm needs to test at least $m_2(b')$ of the variables not in S'.
- Fact 2: To finish Phase 1, Algorithm 1 needs to test at most $m_2(b') + m_3(b')$ additional variables.

To prove Fact 1, note that if $f = j \neq 0$, then during its execution, an optimal algorithm needs to test at least $\lfloor n/2 \rfloor + 1 - N_j(b')$ of the variables not in S'; if $f = 0$, an optimal algorithm needs to test at least $m_1(b')$ variables not in S'. Since

$$m_2(b') = \lceil n/2 \rceil - N_1(b') - \sum_{j=3}^{d} N_j(b') \leq \lfloor n/2 \rfloor + 1 - N_1(b') \leq \lfloor n/2 \rfloor + 1 - N_j(b')$$

and $m_2(b') \leq m_1(b')$, we have shown that Fact 1 holds, regardless of the value of f.

We now show Fact 2. Suppose that after performing $m_2(b') + m_3(b')$ additional tests, Algorithm 1 still hasn't terminated. Let b'' denote the partial assignment after the $m_2(b') + m_3(b')$ additional tests are performed. If $m_2(b'') \neq 0$, the number of 2's among those $m_2(b') + m_3(b')$ variables must be more than $m_3(b')$; if $m_3(b'') \neq 0$, the number of 3's among those variables must be more than $m_2(b')$. Those two conditions cannot occur simultaneously, and if one of $m_2(b''), m_3(b'')$ is not equal to 0, the remaining $m_j(b'')$ for $j > 3$ are definitely 0. If both $m_2(b'') = m_3(b'') = 0$, we can use this argument again for $m_4(b'')$ and $m_5(b'')$. If still not finished, we can use it for $m_6(b'')$ and $m_7(b'')$, etc. Hence, there are at most two $m_j(b'')$ that are not equal to 0 after testing the additional $m_2(b') + m_3(b')$ variables because only $m_1(b'')$ and one other $m_j(b'')$ could be greater than 0. Thus Phase 1 has ended, proving Fact 2.

Having proved the above two facts, which relate to the status of Algorithm 1 when it has not yet completed Phase 1, we now analyze the total cost spent in Phase 1, relative to the total cost spent in an optimal algorithm. Let σ denote the indices of variables such that

$$c_{\sigma_1} \leq c_{\sigma_2} \leq \cdots \leq c_{\sigma_n}.$$

Let k be the number of variables tested by Algorithm 1 in Phase 1. Thus the variables tested in Phase 1 are $\{x_{\sigma_1}, \ldots, x_{\sigma_k}\}$.

Now let b' be the partial assignment right after Algorithm 1 tests $x_{\sigma_1}, \ldots, x_{\sigma_{k-1}}$. We define $\phi_1, \ldots, \phi_d$ such that

$$m_{\phi_1}(b') \geq m_{\phi_2}(b') \geq \cdots \geq m_{\phi_d}(b').$$

Since testing the next variable x_{σ_k} will end Phase 1, $m_{\phi_3}(b') = 1$, and because testing $\{x_{\sigma_1}, ..., x_{\sigma_{k-1}}\}$ does not provide enough information to determine the value of f, an optimal algorithm needs to test at least one variable from $\{x_{\sigma_k}, \ldots, x_{\sigma_n}\}$ during its execution. Hence, we can compare the most expensive variable tested by an optimal algorithm, which costs at least c_{σ_k}, with the last two variables tested by Algorithm 1 in Phase 1, which cost $c_{\sigma_{k-1}} + c_{\sigma_k}$.

Now, suppose b' is the partial assignment right after testing $x_{\sigma_1}, \ldots, x_{\sigma_{k-3}}$, and consider the corresponding $\phi_1, \ldots, \phi_d$. Because there are exactly 3 variables tested in Phase 1 after $\{x_{\sigma_1}, \ldots, x_{\sigma_{k-3}}\}$, and by Fact 2, $m_{\phi_2}(b') + m_{\phi_3}(b')$ is an upper bound on the number of additional variables tested in Phase 1, we have that $3 \leq m_{\phi_2}(b') + m_{\phi_3}(b')$. Since the m_j's are integer-valued functions, we have $m_{\phi_2}(b') \geq 2$. By Fact 1, an optimal algorithm needs to test at least $m_{\phi_2}(b')$ variables not in $\{x_{\sigma_1}, \ldots, x_{\sigma_{k-3}}\}$ during its execution, so it needs to test at least two variables from $\{x_{\sigma_{k-2}}, \ldots, x_{\sigma_n}\}$. Hence, we can compare the second-most expensive variable tested by an optimal algorithm, which costs at least $c_{\sigma_{k-2}}$, with the third-last and the fourth-last variables tested by Algorithm 1 in Phase 1, which cost $c_{\sigma_{k-3}} + c_{\sigma_{k-2}}$.

More generally, let b' denote the partial assignment right after having tested variables $x_{\sigma_1}, \ldots, x_{\sigma_{k-\ell}}$ where $1 \leq \ell \leq k-1$, and again consider the corresponding $\phi_1, \ldots, \phi_d$. Algorithm 1 does ℓ tests in Phase 1 after testing $x_{\sigma_1}, \ldots, x_{\sigma_{k-\ell}}$, where $\ell \leq m_{\phi_2}(b') + m_{\phi_3}(b') \leq 2m_{\phi_2}(b')$ by Fact 1. Applying Fact 2 then yields that an optimal algorithm needs to test at least $m_{\phi_2}(b') \geq \lceil \ell/2 \rceil$ variables from $\{x_{\sigma_{k-\ell+1}}, \ldots, x_{\sigma_n}\}$. Hence, the $\lceil \ell/2 \rceil$th-most expensive variable tested by an optimal algorithm costs at least $c_{\sigma_{k-\ell+1}}$. On the other hand, the ℓth-last and the $(\ell+1)$st-last variable tested by Algorithm 1 in Phase 1 cost $c_{\sigma_{k-\ell+1}} + c_{\sigma_{k-\ell}}$ in total.

To conclude our analysis of Phase 1, the cost Algorithm 1 spends in Phase 1 is upper-bounded by twice the total cost of the $\lceil k/2 \rceil$ most expensive variables tested by an optimal algorithm, and hence it is also upper-bounded by twice the total cost of all variables tested by an optimal algorithm.

After Phase 1, there are three remaining possibilities: $f = \alpha$, $f = \beta$ and $f = 0$. Let OPT denote an optimal adaptive strategy for the problem. If $f = \alpha$, clearly, our algorithm queries the remaining variables optimally; if $f = \beta$, since we always need to prove $f \neq \alpha$, our algorithm spends at most $2E[\mathsf{OPT}]$ in Phase 2; if $f = 0$, we need to show $f \neq \alpha$ and $f \neq \beta$, hence, our algorithm spends at most $2E[\mathsf{OPT}]$ in Phase 2 as well. Combining the cost we spent in Phase 1, Algorithm 1 is a 4-approximation algorithm for evaluating an absolute-majority function when $c_i > 0$. ◀

The SBB strategy is a highly adaptive strategy; it is easy to see that Algorithm 1 may require $\Omega(n)$ rounds of adaptivity. Motivated by this, we also sketch a modification of this algorithm such that the new algorithm needs only 3 rounds of adaptivity, but the approximation ratio is increased slightly from 4 to 6. Specifically, we replace Phase 2 which contains the two runs of the SBB strategy. After Phase 1 is completed, if we are still uncertain about the result of the election, we can first evaluate if α is the absolute-majority winner by using the non-adaptive 2-approximation algorithm for evaluating a k-of-n function, which was introduced in Gkenosis et al. [13]. This algorithm performs a modified (cost-sensitive) round robin between two orderings, one in increasing order of $c_i/p_{i,\alpha}$, and the other in increasing order of $c_i/(1 - p_{i,\alpha})$. The modified round robin is due to Allen et al. [3] and we present the pseudocode in the full version of the paper. It keeps track of the next vote to be inspected in each ordering, and the total cost incurred so far for each ordering. It computes, for each ordering, the sum of the cost incurred so far, plus the cost of the next test. It performs the next test from the ordering for which this sum is smaller (breaking ties arbitrarily). We call this Phase 2.

Since the problem of evaluating f requires determining whether or not α is the winner, the cost spent in Phase 2 is upper-bounded by twice the expected cost of evaluating the value of f. If we know α is not the absolute-majority winner after Phase 2, we start Phase 3 by evaluating if β is the winner using the same algorithm in Gkenosis et al. This will again add at most twice the expected cost of evaluating f. Clearly, if both α and β are not the winner, there is no absolute-majority winner in the election.

We thus have the following theorem.

▶ **Theorem 2.** *There exists a 6-approximation algorithm for evaluating the absolute-majority function with 3 rounds of adaptivity.*

We note that it is possible to reduce the rounds of adaptivity from 3 to 2 using a modified round-robin applied to 4 different non-adaptive orderings. We give details in the full version of the paper.

4 Relative Majority

We now turn our attention to the relative-majority version of our problem and present an 8-approximation algorithm. Similar to our previous approach, we keep a partial assignment b to track the values of inspected votes and also to indicate which vote values are not yet known. To show that a candidate j has more votes than a candidate k is equivalent to showing that

$$N_k(b) + N_*(b) < N_j(b).$$

The fact that $N_*(b) = n - \sum_{k\in\{1,\dots,d\}} N_k(b)$ motivates designing

$$g_{j,k}(b) = \min\left\{N_j(b) + \sum_{\ell\in\{1,\dots,d\}\setminus\{k\}} N_\ell(b), n+1\right\}$$

Algorithm 2 An 8-apx. adaptive algorithm for evaluating the relative-majority function

```
b ← {∗}^n
while the value of f is not certain do
    if there are more than 2 candidates that can possibly win then
        test the next untested variable in increasing order of c_i
        update b
    else
        α ← any j ∈ arg max_{j∈{1,...,d}} N_j(b)
        β ← any j ∈ arg max_{j∈{1,...,d}\{α}} N_j(b)
        evaluate if f = α by using the Adaptive Dual Greedy algorithm (cf. [8])
        if f = α then
            return f = α
        else
            evaluate if f = β by testing remaining variables in increasing order of c_i / p_{i,α}
            if f = β then
                return f = β
            else
                return f = 0
return the value of f
```

such that $g_{j,k}(b) = n + 1$ iff candidate j is guaranteed to have more votes than candidate k. Hence, to conclude that j is a relative-majority winner, we must have $g_{j,k}(b) = n + 1$ for all $k \in \{1, \ldots, d\} \setminus \{j\}$. Our algorithm again has two phases. In Phase 1, we still inspect the votes in increasing order of their costs. We end Phase 1 when there are at most two candidates who can possibly win and let α, β denote the two candidates. Then, there are at most three possible results of the election: α wins, β wins, or it is a tie. We will first determine if α wins or not. Recall that, if α wins, they must have more votes than β, which means $g_{\alpha,\beta}(b)$ needs to be at least $n + 1$ at some point. Since having a vote for α,β, or any candidate in $\{1, \ldots, d\} \setminus \{\alpha, \beta\}$ contributes 2, 0, or 1 to $g_{\alpha,\beta}$, respectively, we can use the Adaptive Dual Greedy algorithm introduced by Deshpande et al. [8] to evaluate if $g_{\alpha,\beta}(b)$ can ever exceed the threshold $n + 1$. If yes, clearly α is the winner; if not, then we again use the SBB strategy to decide if β is the winner or it is a tie. The pseudocode is presented in Algorithm 2 (again, this is slightly more verbose than necessary for readability).

We now give an analysis of the algorithm.

▶ **Lemma 3.** *The cost that Algorithm 2 spends in Phase 1 is at most* 4 *times the cost of an optimal algorithm.*

Proof. For each $j \in \{1, \ldots, d\}$ and $k \in \{1, \ldots, d\} \setminus \{j\}$, we define $m_{j,k}(b)$ to be the distance from $g_{j,k}(b)$ to $n + 1$, so we have

$$m_{j,k}(b) = n + 1 - g_{j,k}(b),$$

and define $M_j(b)$ such that

$$M_j(b) = \max_{k \in \{1,\ldots,d\} \setminus \{j\}} m_{j,k}(b),$$

which is the largest distance among the $d - 1$ utility functions $g_{j,k}$ of candidate j to $n + 1$.

Recall the definition of the relative-majority function f. We can see that, when $f = 0$, which means it is a draw situation, clearly we need to test all variables. When there exists a

relative-majority winner, let b' be the partial assignment at any moment before Algorithm 2 terminates, and let j^* be an arbitrary element in the set $\arg\min_{j\in\{1,\dots,d\}} M_j(b')$. Then, any optimal algorithm needs to test at least $\lceil M_{j^*}(b')/2\rceil$ variables from the untested variables since any single test can contribute at most 2 to any $g_{j,k}$. Here we note that $M_{j^*}(b') > 0$, otherwise j^* would have strictly more votes than all other candidates, and we can conclude $f = j^*$.

Now we define $\gamma, \delta \in \{1,\dots,d\} \setminus \{j^*\}$ such that

$$\forall k \in \{1,\dots,d\} \setminus \{\gamma,\delta,j^*\}, \quad M_{j^*}(b') = m_{j^*,\gamma}(b') \geq m_{j^*,\delta}(b') \geq m_{j^*,k}(b'),$$

which implies that

$$\forall k \in \{1,\dots,d\} \setminus \{\gamma,\delta,j^*\}, \quad N_\gamma(b') \leq N_\delta(b') \leq N_k(b')$$

since $m_{j^*,k}(b') = n+1-g_{j^*,k}(b')$ for any $k \in \{1,\dots,d\}\setminus\{j^*\}$.

Suppose Phase 1 has not ended, meaning that there are more than two candidates who can still win at this moment. Then, we claim that testing any $m_{j^*,\gamma}(b') + m_{j^*,\delta}(b')$ additional variables will end Phase 1.

We now prove this claim. Let $m_\gamma = m_{j^*,\gamma}(b')$, $m_\delta = m_{j^*,\delta}(b')$. Let b'' denote the partial assignment after testing any $m_\gamma + m_\delta$ additional variables. Suppose there exists a $\mu \in \{1,\dots,d\}\setminus\{j^*\}$ such that $m_{j^*,\mu}(b'') > 0$. Then

$$\begin{aligned}
& m_{j^*,\mu}(b'') > 0 \\
\Leftrightarrow\ & m_{j^*,\mu}(b') - (N_{j^*}(b'') - N_{j^*}(b')) - m_\gamma - m_\delta + (N_\mu(b'') - N_\mu(b')) > 0 \\
\Leftrightarrow\ & N_\mu(b'') - N_\mu(b') > m_\gamma + m_\delta + N_{j^*}(b'') - N_{j^*}(b') - m_{j^*,\mu}(b'') \\
\Leftrightarrow\ & N_\mu(b'') - N_\mu(b') > (n+1-g_{j^*,\gamma}(b')) + (n+1-g_{j^*,\delta}(b')) \\
& \qquad\qquad + N_{j^*}(b'') - N_{j^*}(b') - (n+1-g_{j^*,\mu}(b')) \\
\Leftrightarrow\ & N_\mu(b'') - N_\mu(b') > (n+1) - g_{j^*,\gamma}(b') - g_{j^*,\delta}(b') + g_{j^*,\mu}(b') + N_{j^*}(b'') - N_{j^*}(b').
\end{aligned}$$

Similarly, if there exists a $\nu \in \{1,\dots,d\}\setminus\{j^*,\mu\}$ such that $m_{j^*,\nu}(b'') > 0$, we have

$$N_\nu(b'') - N_\nu(b') > (n+1) - g_{j^*,\gamma}(b') - g_{j^*,\delta}(b') + g_{j^*,\nu}(b') + N_{j^*}(b'') - N_{j^*}(b').$$

Then, we can see that if both $m_{j^*,\mu}(b'') > 0$ and $m_{j^*,\nu}(b'') > 0$ hold, we have

$$\begin{aligned}
& N_\mu(b'') - N_\mu(b') + N_\nu(b'') - N_\nu(b') \\
&> 2(n+1-g_{j^*,\gamma}(b') - g_{j^*,\delta}(b')) + g_{j^*,\mu}(b') + g_{j^*,\nu}(b') + 2(N_{j^*}(b'') - N_{j^*}(b')) \\
&\geq 2(n+1) - g_{j^*,\gamma}(b') - g_{j^*,\delta}(b') + 2(N_{j^*}(b'') - N_{j^*}(b')) \\
&\geq m_\gamma + m_\delta,
\end{aligned}$$

where the second inequality comes from $g_{j^*,\mu}(b') + g_{j^*,\nu}(b') \geq g_{j^*,\gamma}(b') + g_{j^*,\delta}(b')$ and the third inequality comes from $N_{j^*}(b'') - N_{j^*}(b') \geq 0$. We have thus shown that $N_\mu(b'') - N_\mu(b') + N_\nu(b'') - N_\nu(b') > m_\gamma + m_\delta$, which is a contradiction because it implies that more than $m_\gamma + m_\delta$ additional votes were inspected. Therefore, there is at most one k such that $m_{j^*,k}(b'') > 0$. This proves the claim that inspecting any $m_\gamma + m_\delta$ additional votes will end Phase 1.

By definition, $m_{j^*,k}(b'') = 0$ is equivalent to $g_{j^*,k}(b'') = n+1$, which means j^* has strictly more votes than k. This eliminates the possibility that k is the winner. Since there exists at most one k^* that satisfies $m_{j^*,k^*} \neq 0$, there are at most two candidates, j^* and k^*, who can possibly win.

Now we can use a similar argument as in the proof of Theorem 1 to prove the rest of the lemma. Let $\sigma_1, \ldots, \sigma_n$ denote the indices of variables such that $c_{\sigma_1} \leq c_{\sigma_2} \leq \cdots \leq c_{\sigma_n}$. In an arbitrary realization, we assume at the time Phase 1 ends, Algorithm 2 has tested $x_{\sigma_1}, \ldots, x_{\sigma_k}$.

Let b' denote the partial assignment when we have obtained the values of $x_{\sigma_1}, \ldots, x_{\sigma_{k-\ell}}$ where $\ell \geq 1$. From our previous claim, we know that

$$\ell \leq m_{j^*,\gamma}(b') + m_{j^*,\delta}(b') \leq 2 \cdot m_{j^*,\gamma}(b').$$

On the other hand, we know that an optimal algorithm needs to test at least $\lceil m_{j^*,\gamma}(b')/2 \rceil$ variables outside of $\{x_{\sigma_1}, \ldots, x_{\sigma_{k-\ell}}\}$, which implies that an optimal algorithm needs to test at least $\lceil \ell/4 \rceil$ variables from $\{x_{\sigma_{k-\ell+1}}, \ldots, x_n\}$. Hence, for example, by setting $\ell = 1$, we can compare the most expensive variable tested by an optimal algorithm, which costs at least c_{σ_k}, with the total costs of $\{x_{\sigma_{k-3}}, x_{\sigma_{k-2}}, x_{\sigma_{k-1}}, x_{\sigma_k}\}$; by setting $\ell = 5$, we can compare the second-most expensive variable tested by an optimal algorithm, which costs at least $c_{\sigma_{k-4}}$, with the total costs of $\{x_{\sigma_{k-7}}, x_{\sigma_{k-6}}, x_{\sigma_{k-5}}, x_{\sigma_{k-4}}\}$. Generalizing these observations, we can see that the $\lceil \ell/4 \rceil$th-most expensive variable tested by any optimal algorithm costs at least $c_{\sigma_{k-\ell+1}}$. Hence, the total cost that Algorithm 2 spends in Phase 1, $c_{\sigma_1} + \cdots + c_{\sigma_k}$, is at most 4 times the cost of an optimal algorithm. ◀

▶ **Theorem 4.** *Algorithm 2 is an adaptive 8-approximation algorithm for evaluating the relative-majority function.*

Proof. Let OPT denote an optimal algorithm. By using Lemma 3, we know that Algorithm 2 spent at most $4E[\mathsf{OPT}]$ in Phase 1. Hence, we just need to solve the induced problem after Phase 1.

Let α, β denote the two candidates who can possibly win. We know that evaluating function f will also evaluate if $f = \alpha$. Hence, the expected cost of an optimal algorithm for evaluating if α wins is upper-bounded by the expected cost of an optimal algorithm for evaluating the result of the election. Since there are only two candidates α, β remaining, evaluating if $f = \alpha$ is equivalent to evaluating if α has more votes than β. This problem can be understood as evaluating a variant of a linear threshold formula that was studied by Deshpande et al. [8]. In particular, let b' denote the partial assignment right after we figured α, β, we need to determine whether or not the following inequality holds:

$$N_\alpha(b') + \left(\sum_{k \in \{1,\ldots,d\} \setminus \{\beta\}} N_k(b') \right) + \sum_{i : b'_i = *} y_i \geq n + 1.$$

Here $y_i = 2$ if we find $x_i = \alpha$; $y_i = 1$ if $x_i \in \{1, \ldots, d\} \setminus \{\alpha, \beta\}$; and $y_i = 0$ if $x_i = \beta$. This is a linear threshold evaluation problem, involving ternary rather than binary variables. The Adaptive Dual Greedy algorithm of Deshpande et al. can be used to solve this ternary linear threshold function evaluation problem, and determine whether $f = \alpha$. A slight modification of the analysis used by Deshpande et al. shows that it spends at most $3E[\mathsf{OPT}]$. We defer details of the use of Adaptive Dual Greedy and its approximation bound to the full version of the paper.

Suppose $f \neq \alpha$, what remains is to decide if $f = 0$ or $f = \beta$. Recall that at the point we know $f \neq \alpha$, let b' be the partial assignment, we have

$$N_\alpha(b') + N_*(b') \leq N_\beta(b').$$

This means that, when the equality does not hold, we know $f = \beta$, since even if all the remaining variables have value α, β will have strictly more votes. Therefore, the induced

problem can be considered as evaluating a conjunction (i.e., checking whether $x_i = \alpha$ for all untested x_i's), which can be solved optimally by using the SBB strategy that tests the remaining variables in increasing order of $c_i/(1 - p_{i,\alpha})$.

Hence, Algorithm 2 spends at most $4E[\mathsf{OPT}] + 3E[\mathsf{OPT}] + E[\mathsf{OPT}] = 8E[\mathsf{OPT}]$. ◀

5 Conclusion

In this paper, we have introduced the problem of quickly determining the winner of an election. We have also given the first constant-factor approximation algorithms for this problem in both the absolute-majority and the relative-majority case. While the approximation guarantees are one-digit numbers, we have no reason to believe that they match some type of approximation hardness. In fact, even NP-hardness remains an open problem. Assuming our problem is NP-hard, proving this is the case might require new techniques. Some stochastic evaluation problems are known to be NP-hard, for example, evaluation of Boolean linear threshold functions (cf. [20]) and evaluation of s-t connectivity in general graphs [9]. However, it is open whether a number of Stochastic Score function evaluation problems, related to our problem, are NP-hard [1, 14, 16]. The question of whether the stochastic evaluation problem for Boolean read-once formulas is NP-hard has been open since the 1970's (cf. [20]).

A structural question left open by our work is whether the adaptivity gap of our problem is constant. The strongest lower bound on the adaptivity gap that we are aware of comes from the lower bound of 1.5 for k-of-n functions [18]. Extending the corresponding construction, one may try to consider an instance in which there are some votes of negligible cost that determine a forerunner, who can then be proved to be a winner cheaply by an adaptive strategy (by selecting the corresponding votes) but not by a non-adaptive strategy (because it does not know which votes to inspect). The extension is, however, not straightforward. One is restricted to certain distributions of who is the forerunner, and even votes for another candidate make progress on goal functions associated with other candidates.

Many natural extensions and variants of our model are possible, e.g., to other voting systems. A relatively small extension would concern larger target numbers of votes than $n/2$, possibly even candidate-specific ones. Such scenarios can be approached by adding an appropriate number of 0-cost votes for each candidate (i.e., with probability close enough to 1) and then using our algorithm for the absolute-majority version. Similarly, one could consider weighted voting systems (with modern applications in liquid democracy or voting in shareholder meetings). Ranked-voting systems could also be investigated.

References

1 Jayadev Acharya, Ashkan Jafarpour, and Alon Orlitsky. Expected query complexity of symmetric Boolean functions. In *Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 26–29, 2011.

2 Arpit Agarwal, Sepehr Assadi, and Sanjeev Khanna. Stochastic submodular cover with limited adaptivity. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 323–342, 2019.

3 Sarah R Allen, Lisa Hellerstein, Devorah Kletenik, and Tonguç Ünlüyurt. Evaluation of monotone dnf formulas. *Algorithmica*, 77(3):661–685, 2017.

4 Yosi Ben-Dov. Optimal testing procedures for special structures of coherent systems. *Management Science*, 27(12):1410–1420, 1981.

5 Guy Blanc, Jane Lange, and Li-Yang Tan. Query strategies for priced information, revisited. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1638–1650, 2021.

6 Katalin Bognar, Tilman Börgers, and Moritz Meyer-ter Vehn. An optimal voting procedure when voting is costly. *Journal of Economic Theory*, 159:1056–1073, 2015.

7 Moses Charikar, Ronald Fagin, Venkatesan Guruswami, Jon Kleinberg, Prabhakar Raghavan, and Amit Sahai. Query strategies for priced information. In *ACM Symposium on Theory of Computing (STOC)*, pages 582–591, 2000.

8 Amol Deshpande, Lisa Hellerstein, and Devorah Kletenik. Approximation algorithms for stochastic submodular set cover with applications to Boolean function evaluation and min-knapsack. *ACM Transactions on Algorithms (TALG)*, 12(3):1–28, 2016.

9 Luoyi Fu, Xinzhe Fu, Zhiying Xu, Qianyang Peng, Xinbing Wang, and Songwu Lu. Determining source–destination connectivity in uncertain networks: Modeling and solutions. *IEEE/ACM Transactions on Networking*, 25(6):3237–3252, 2017.

10 Alex Gershkov and Balázs Szentes. Optimal voting schemes with costly information acquisition. *Journal of Economic Theory*, 144(1):36–68, 2009.

11 Rohan Ghuge, Anupam Gupta, and Viswanath Nagarajan. The power of adaptivity for stochastic submodular cover. In *International Conference on Machine Learning (ICML)*, pages 3702–3712, 2021.

12 Rohan Ghuge, Anupam Gupta, and Viswanath Nagarajan. Non-adaptive stochastic score classification and explainable halfspace evaluation. In *International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 277–290, 2022.

13 Dimitrios Gkenosis, Nathaniel Grammel, Lisa Hellerstein, and Devorah Kletenik. The stochastic score classification problem. In *European Symposium on Algorithms (ESA)*, pages 36:1–36:14, 2018.

14 Dimitrios Gkenosis, Nathaniel Grammel, Lisa Hellerstein, and Devorah Kletenik. The stochastic Boolean function evaluation problem for symmetric Boolean functions. *Discrete Applied Mathematics*, 309:269–277, 2022.

15 Michel Goemans and Jan Vondrák. Stochastic covering and adaptivity. In *Latin American Theoretical Informatics Symposium (LATIN)*, pages 532–543, 2006.

16 Nathaniel Grammel, Lisa Hellerstein, Devorah Kletenik, and Naifeng Liu. Algorithms for the unit-cost stochastic score classification problem. *Algorithmica*, 84(10):3054–3074, 2022.

17 Naifeng Liu. Two 6-approximation algorithms for the stochastic score classification problem. *CoRR*, abs/2212.02370, 2022. `arXiv:2212.02370`, `doi:10.48550/arXiv.2212.02370`.

18 Benedikt M. Plank and Kevin Schewior. Simple algorithms for stochastic score classification with small approximation ratios. *CoRR*, abs/2211.14082, 2022. `arXiv:2211.14082`, `doi:10.48550/arXiv.2211.14082`.

19 Salam Salloum and Melvin A Breuer. An optimum testing algorithm for some symmetric coherent systems. *Journal of Mathematical Analysis and Applications*, 101(1):170–194, 1984.

20 Tonguç Ünlüyurt. Sequential testing of complex systems: a review. *Discrete Applied Mathematics*, 142(1-3):189–205, 2004.