Fishing out Winners from Vote Streams

Arnab Bhattacharyya and Palash Dey

Indian Institute of Science, Bangalore

{arnabb,palash}@csa.iisc.ernet.in

Abstract

We investigate the problem of winner determination from computational social choice theory in the data stream model. Specifically, we consider the task of summarizing an arbitrarily ordered stream of n votes on m candidates into a small space data structure so as to be able to obtain the winner determined by popular voting rules. As we show, finding the exact winner requires storing essentially all the votes. So, we focus on the problem of finding an ε-*winner*, a candidate who could win by a change of at most ε fraction of the votes. We show non-trivial upper and lower bounds on the space complexity of ε-winner determination for several voting rules, including k-approval, k-veto, scoring rules, approval, maximin, Bucklin, Copeland, and plurality with run off.

1   Introduction

A common and natural way to aggregate preferences of agents is through an *election*. In a typical election, we have a set of m candidates and a set of n voters, and each voter reports his ranking of the candidates in the form of a *vote*. A *voting rule* selects one candidate as the winner once all voters provide their votes. Determining the winner of an election is one of the most fundamental problems in social choice theory.

We consider elections held in an online setting where voters vote in arbitrary order, and we would like to find the winner at any point in time. A very natural scenario where this occurs is an election conducted over the Internet. For instance, websites often ask for rankings of restaurants in a city and would like to keep track of the "best" restaurant according to some fixed voting rule. Traditionally, social choice theory addresses settings where the number of candidates is much smaller than the number of voters. However, we now often have situations where both the candidate set and voter set are very large. For example, the votes may be the result of high-frequency measurements made by sensors in a network [26], and a voting rule could be used to aggregate the measurements (as argued in [7]). Also, in online participatory democracy systems, such as [wid, syn], the number of candidates can be as large as the number of voters. The naïve way to conduct an online election is to store all the vote counts in a database and recompute the winner whenever it is needed. The space complexity of this approach becomes infeasible if the number of candidates or the number of votes is too large. Can we do better? Is it possible to compress the votes into a short summary that still allows for efficient recovery of the winner?

This question can be naturally formulated in the *data stream model* [3, 20]. Votes are interpreted as items in a data stream, and the goal is to devise an algorithm with minimum space requirement to determine the election winner. In the simplest setting of the plurality voting

rule, where each vote is simply an approval for a single candidate and the winner is the one who is approved by the most, our problem is closely related to the classic problem of finding heavy hitters [9, 12] in a stream. For other popular voting rules, such as *Borda, Bucklin* or *Condorcet consistent* voting rules, the questions become somewhat different.

Regardless of the voting rule, if the goal is to recover only the winner and the stream of votes is arbitrary, then it becomes essentially impossible to do anything better than the above-mentioned naïve solution (even when the algorithm is allowed to be randomized). Although we prove this formally, the reason should be intuitively clear: the winner may be winning by a very tiny margin thereby making every vote significant to the final outcome. We therefore consider a natural relaxation of the winner determination problem, where the algorithm is allowed to output any candidate who could have been the winner, according to the voting rule under consideration, by a change of at most εn votes. We call such a candidate an ε-*winner*; similar notions were introduced in [25, 36]. Note that if the winner wins by a *margin of victory* [36] of more than εn, there is a unique ε-winner.

In this work, we study streaming algorithms to solve the (ε, δ)-WINNER DETERMINATION problem, i.e. the task of determining, with probability at least $1 - \delta$, an ε-winner of any given vote stream according to popular voting rules. Our algorithms are necessarily randomized.

1.1 Our Contributions

We initiate the study of streaming algorithms for the (ε, δ)–WINNER DETERMINATION problem with respect to various voting rules. The results for the (ε, δ)–WINNER DETERMINATION problem, when both ε and δ are positive, are summarized in Table 1. (When ε or δ equals 0, we prove that the space requirements are much larger.)

We also exhibit algorithms, having space complexity nearly same as Table 1, for the more general *sliding window* model, introduced by Datar et. al. in [13]. In this setting, for some parameter N, we want to find an ε-winner with respect to the N most recent votes in the stream, clearly a very well motivated scenario in online elections.

1.2 Related Work

1.2.1 Social Choice

To the best of our knowledge, our work is the first to systematically study the approximate winner determination problem in the data stream model. A conceptually related work is that of Conitzer and Sandholm [10] who study the communication complexity of common voting rules. They consider n parties each of whom knows only their own vote but, through a communication protocol, would like to compute the winner according to a specific voting rule. Observe that a streaming algorithm for exact winner determination using s bits of memory space immediately[I] implies a one-way communication protocol where each party transmits s bits. However, it turns out that their results only imply weak lower bounds for the space complexity of streaming algorithms. Moreover, [10] does not study determination of ε-winners. The communication complexity of voting rules was also highlighted by Caragiannis and Procaccia in [7].

In a recent work, we [15] studied the problem of determining election winners from a random sample of the vote distribution. Since we can randomly sample from a stream of votes using a small amount of extra storage, the bounds from [15] are also useful in the streaming context.

[I]Each party can input its vote into the stream and then communicate the memory contents of the streaming algorithm to the next party.

Voting Rule	Space complexity	
	Upper bound	Lower bound
Generalized plurality[‡]	$O\big(\min\{\frac{1}{\varepsilon}\log m, m\log\frac{1}{\varepsilon}\} + \log\log n\big)$ [Theorem 3 and 4]	$\Omega(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon} + \frac{1}{\sqrt{\varepsilon}}\log m + \log\log n)$ if $m \geq \frac{1}{\varepsilon}$[†] [Theorem 18]
k-veto[⋆]	$O(\min\{\frac{k}{\varepsilon}\log m, m\log\frac{\log(m-k+1)}{\varepsilon}\} + \log\log n)$ [Theorem 3]	$\Omega(\frac{1}{\varepsilon^\mu}\log\frac{1}{\varepsilon} + \log m + \log\log n)$ if $m \geq \frac{1}{\varepsilon}$[†] for every $\mu \in [0,1)$ [Theorem 16]
Plurality	$O\big(\min\{\frac{1}{\varepsilon}\log m, m\log\frac{1}{\varepsilon}\} + \log\log n\big)$ [Theorem 3 and 4]	
k-approval[⋆]	$O(\min\{\frac{k}{\varepsilon}\log m, m\log\frac{\log(k)}{\varepsilon}\} + \log\log n)$ [Theorem 3]	
Scoring rules	$O(m(\log\log m + \log\frac{1}{\varepsilon}) + \log\log n)$ [Theorem 5]	$\begin{cases}\Omega(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon} + \log m + \log\log n) & \text{if } m \geq \frac{1}{\varepsilon}, \\ \Omega(m\log\frac{1}{\varepsilon} + \log\log n) & \text{if } m \leq \frac{1}{\varepsilon},\end{cases}$
Approval	$O(m(\log\log m + \log\frac{1}{\varepsilon}) + \log\log n)$ [Theorem 6]	[Theorem 15 and 19] [Theorem 20] [Observation 2]
Maximin, Bucklin, Run off	$O(\min\{m^2(\log\log m + \log\frac{1}{\varepsilon}), \frac{1}{\varepsilon^2}m\log^2 m\} + \log\log n)$ [Theorem 7 and 8]	
Copeland	$O(\min\{m^2(\log\log m + \log\frac{1}{\varepsilon}), \frac{1}{\varepsilon^2}m\log^4 m\} + \log\log n)$ [Theorem 7 and 8]	

Table 1: Space complexity for the (ε, δ)-WINNER DETERMINATION problem for various voting rules. We do not show dependence on δ in the table for sake of clarity. $\star$: The lower bound results for the k-approval and k-veto voting rules apply only for $k = O(m^\gamma)$, for every $\gamma \in [0,1)$. †: For the case $m \leq \frac{1}{\varepsilon}$, the lower bound is same as that of other rules. ‡ : Here, each voter has the choice to either approve or disapprove of one candidate and the candidate who has the maximum number of approvals minus disapprovals wins.

In that work, the goal was to find the winner who was assumed to have a *margin of victory* [36] of at least ε, but the same arguments also work for finding ε-winners.

1.2.2 Streaming

The field of streaming algorithms has been the subject of intense research over the past two decades in both the algorithms and database communities. The theoretical foundations for the area were laid by [3, 20]. A stream is a sequence of data items $\sigma_1, \sigma_2, \ldots, \sigma_n$, drawn from the universe $[m]$, such that on each *pass* through the stream, the items are read once in that order. The frequency vector associated with the stream $f = (f_1, \cdots, f_m) \in \mathbb{Z}^m$ is defined as f_j being the number of times j occurs as an item in the stream. In this definition, the stream is *insertion-only*; more generally, in the *turnstile model*, items can both be inserted and deleted from the stream, in which case the frequency vector maintains the cumulative count of each element in $[m]$. General surveys of the area can be found in [32, 33].

Algorithms for the insertion-only case were discovered before the formulation of the data streaming model. Consider the *point-query* problem: for a stream of n items from a universe of size m and a parameter $\varepsilon > 0$, the goal is to output, for any item $j \in [m]$, an estimate $\hat{f}_j$ such that $|\hat{f}_j - f_j| \leq \varepsilon n$. Misra and Gries [30] gave[II] an elegant but simple deterministic algorithm requiring only $O(\min\{m, 1/\varepsilon\} \cdot (\log m + \log n))$ space in bit complexity. Since to find an ε-winner

---

[II]The algorithm can be viewed as a generalization of the Boyer-Moore [5, 17] algorithm for $\varepsilon = 1/2$. It was also rediscovered 20 years later by [14, 22].

for the plurality voting rule, it's enough to solve the point query problem and output the j with maximum $\hat{f}_j$, Misra-Gries automatically implies $O(\min\{m, 1/\varepsilon\} \cdot (\log m + \log n))$ space complexity for plurality. We use sampling to improve the dependence on n and prove tightness in terms of ε and n. Our algorithms for many of the other voting rules are also based on the Misra-Gries algorithm. We note that in place of Misra-Gries, there are several other deterministic algorithms which could have been used, such as *Lossy Counting* [27] and *Space Saving* [28], but they would not change the asymptotic space complexity bounds. A thorough overview of the point query, or frequency estimation, problem can be found in [11].

For the more general turnstile model, the point query problem for such streams is that of finding $\hat{f}_j$, for every j, such that $|\hat{f}_j - f_j| \le \varepsilon\|f\|_1$. The best result for this problem is due to Cormode and Muthukrishnan, the randomized *count-min sketch* [12], which has space complexity $O(\frac{1}{\varepsilon} \log m \log n)$ in bits. The space bound was proved to be essentially tight by Jowhari et al. in [21]. In our context, the stream is a sequence of votes; so, our problems are mostly, just by definition, insertion-only. However, the count-min sketch becomes useful in our applications (i) if voters can issue retractions of their votes, and (ii) to maintain counts of random samples drawn from streams of unknown length.

1.3 Technical Overview

Upper Bounds. The streaming algorithms that achieve the upper bounds shown in Table 1 are obtained through applying frequency estimation algorithms, such as Misra-Gries or count-min sketch, appropriately on a subsampled stream. The number of samples needed to obain ε-winners for the various voting rules was previously analyzed in [15].

Lower Bounds. Our main technical novelty is in the proofs of the lower bounds for the (ε, δ)-winner determination problem. Usually, in the "heavy hitters" problem in the algorithms literature, the task is *roughly* to determine the set of items with frequency above εn. Since there can be $1/\varepsilon$ such items, a space lower bound of $\log \binom{m}{1/\varepsilon} = \Omega(\frac{1}{\varepsilon} \log(\varepsilon m))$ immediately follows for $m \gg 1/\varepsilon$. In contrast, we wish to determine only one ε-winner, so that just $\log m$ bits are needed to output the result. In order to obtain stronger lower bounds that depend on ε, we need to resort to other techniques. Moreover, note that our lower bounds are in the insertion-only stream model, whereas previous lower bounds for frequency estimation problems are usually for the more general turnstile model.

We prove these bounds through new reductions from fundamental problems in communication complexity. To give a flavor of the reductions, let us sketch the proof for the plurality voting rule. Consider each additive term separately in the lower bound.

- **$\log \log n$**: Suppose Alice has a number $1 \le a \le n$ and Bob a number $1 \le b \le n$, and Bob wishes to know whether $a > b$ through a protocol where communication is one way from Alice to Bob. It is known [29, 34] that Alice is required to send $\Omega(\log n)$ bits to Bob. We can reduce this problem to finding a $1/3$-winner in a plurality election among two candidates by having Alice push 2^a approvals for candidate 1 into the stream and Bob pushing 2^b approvals for candidate 2; the $\Omega(\log \log n)$ lower bound follows.

- **$(1/\varepsilon) \log(1/\varepsilon)$ when $m \ge 1/\varepsilon$**: Consider the INDEXING problem over an arbitrary alphabet: Alice has a vector $x \in [t]^m$ and Bob an index $i \in [m]$, and Bob wants to find x_i through a one-way protocol from Alice to Bob. Ergün et al [16], extending [29]'s proof for the case of $t = 2$, show Alice needs to send $\Omega(m \log t)$ bits. For $t = m = 1/\sqrt{\varepsilon}$, we reduce INDEXING to ε-winner determination for a plurality election. Let the candidate set be $[t] \times [m]$. Alice (given her input x) pushes $n/2$ votes into the stream with $\sqrt{\varepsilon}n/2$ votes to each (x_j, j) for all $j \in [m]$ and

sends over the memory content of the streaming algorithm to Bob who (given his input i) pushes another $n/2$ votes into the stream with $\sqrt{\varepsilon}n/2$ votes to each (a,i) for all $a \in [t]$. Note that candidate (x_i, i) is the unique $\sqrt{\varepsilon}/4$-winner of this plurality election! Using [16]'s lower bound $\Omega(1/\sqrt{\varepsilon}\log(1/\varepsilon))$ on the communication complexity of the INDEXING problem yields our result.

- $m\log(1/\varepsilon)$ **when** $m \leq 1/\varepsilon$: Suppose Alice has a vector $a \in [t]^m$ and Bob a vector $b \in [t]^m$, and Bob wants to find[III] $i = \arg\max_j(a_j + b_j)$ through a one-way protocol. We show by reducing from the AUGMENTED INDEXING problem [16, 29] that Alice needs to send $\Omega(m\log t)$ bits to Bob. Suppose $t = 1/\varepsilon$. Alice imagines her vector a as being the vote count for a plurality election among m candidates, streams in a and runs the streaming algorithm for the problem, and passes the memory output to Bob who also streams in his vector b. The maximum entry in $a + b$ corresponds to a candidate winning by margin at least $\varepsilon^2 n$, hence yielding the $\Omega(m\log(1/\varepsilon))$ lower bound.

2 Preliminaries

2.1 Voting and Voting Rules

Let $\mathcal{V} = \{v_1, \ldots, v_n\}$ be the set of all *voters* and $\mathcal{C} = \{c_1, \ldots, c_m\}$ the set of all *candidates*. If not mentioned otherwise, $\mathcal{V}$, $\mathcal{C}$, n and m denote set of voters, the set of candidates, the number of voters and the number of candidates respectively. Each voter v_i's *vote* is a complete order $\succ_i$ over the candidate set $\mathcal{C}$. For example, for two candidates a and b, $a \succ_i b$ means that the voter v_i prefers a to b. We denote the set of all complete orders over $\mathcal{C}$ by $\mathcal{L}(\mathcal{C})$. Hence, $\mathcal{L}(\mathcal{C})^n$ denotes the set of all n-voters' preference profiles $(\succ_1, \ldots, \succ_n)$. A map $r : \uplus_{n,|\mathcal{C}| \in \mathbb{N}^+} \mathcal{L}(\mathcal{C})^n \longrightarrow 2^{\mathcal{C}}$ is called a *voting rule*. Given a vote profile $\succ \in \mathcal{L}(\mathcal{C})^n$, we call the candidates in $r(\succ)$ the *winners*. Given an election $\mathcal{E} = (\mathcal{V}, \mathcal{C})$, we can construct a weighted graph $\mathcal{G}_\mathcal{E}$, called *weighted majority graph*, from $\mathcal{E}$. The set of vertices in $\mathcal{G}_\mathcal{E}$ is the set of candidates in $\mathcal{E}$. For any two candidates x and y, the weight on the edge (x,y) is $D_\mathcal{E}(x,y) = N_\mathcal{E}(x,y) - N_\mathcal{E}(y,x)$, where $N_\mathcal{E}(x,y)$ (respectively $N_\mathcal{E}(y,x)$) is the number of voters who prefer x to y (respectively y to x). A candidate x is called the *Condorcet winner* in an election $\mathcal{E}$ if $D_\mathcal{E}(x,y) > 0$ for every other candidate $y \neq x$. A voting rule is called *Condorcet consistent* if it selects the Condorcet winner as the winner of the election whenever it exists. Some examples of common voting rules are:

- **Positional scoring rules:** A collection of m-dimensional vectors $\vec{s}_m = (\alpha_1, \alpha_2, \ldots, \alpha_m) \in \mathbb{R}^m$ with $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_m$ and $\alpha_1 > \alpha_m$ for every $m \in \mathbb{N}$ naturally defines a voting rule – a candidate gets score α_i from a vote if it is placed at the ith position. The score of a candidate is the sum of the scores it receives from all the votes. The winners are the candidate with maximum score. The vector α that is 1 in the first k coordinates and 0 in other coordinates gives the k-approval voting rule. The vector α that is 1 in the last k coordinates and 0 in other coordinates is called k-veto voting rule. Observe that the score of a candidate in the k-approval (respectively k-veto) voting rule is the number of approvals (and respectively vetoes) that the candidate receives. 1-approval is called the *plurality* voting rule, and 1-veto is called the *veto* voting rule. The score vector $(m-1, m-2, \ldots, 1, 0)$ gives the *Borda* rule.

- **Generalized plurality:** In generalized plurality voting, each voter approves or disapprove one candidate. The score of a candidate is the number of approvals it receives minus number of disapprovals it receives. The candidates with highest score are the winners. We introduce this rule and consider it to be interesting particularly in an online setting where every voter

---

[III]Assume the maximum is unique.

either likes or dislikes an item; hence each vote is either an approval for a candidate or a disapproval for a candidate.

- **Approval:** In approval voting, each voter approves a subset of candidates. The winners are the candidates which are approved by the maximum number of voters.

- **Maximin:** The maximin score of a candidate x is $\min_{y \neq x} D_{\mathcal{E}}(x, y)$. The winners are the candidates with maximum maximin score.

- **Copeland:** The Copeland score of a candidate x is $|\{y \neq x : D_{\mathcal{E}}(x, y) > 0\}|$. The winners are the candidates with maximum Copeland score.

- **Bucklin:** A candidate x's Bucklin score is the minimum number ℓ such that more than half of the voters rank x in their first ℓ positions. The winners are the candidates with lowest Bucklin score.

- **Plurality with runoff:** The top two candidates according to plurality score are selected first. The pairwise winner of these two candidates is selected as the winner of the election. This rule is often called the *runoff* voting rule.

Among the above, only the maximin and Copeland rules are Condorcet consistent.

2.2 Model of Input Data

In the basic model, the input data is an insertion only stream of elements from some universe $\mathcal{U}$. We note that, in the context of voting in an online scenario, the natural model of input data is the insertion only streaming model over the universe of all possible votes $\mathcal{L}(\mathcal{C})$. The basic model can be generalized to the more sophisticated *sliding window model* where the only *active* items are the last n items, for some parameter n. In this work, we focus on winner determination algorithms for insertion only stream of votes in both basic and sliding window models. The basic input model can also be generalized to another input model, called *turnstile model*, where the input data is a sequence from $\mathcal{U} \times \{1, -1\}$; every element in the stream corresponds to either a unit increment or a unit decrement of frequency of some element from $\mathcal{U}$. We will use the turnstile streaming model (over some different universe) only to design efficient winner determination algorithms for the insertion only stream of votes. We note that, the algorithms for the streaming data can make only one pass over the input data. These one pass algorithms are also called *streaming algorithms*.

2.3 Communication Complexity

We will use lower bounds on *communication complexity* of certain functions to prove space complexity lower bounds for our problems. Communication complexity of a function measures the number of bits that need to be exchanged between two players to compute a function whose input is split among those two players [37]. In a more restrictive *one-way communication model*, the first player sends only one message to the second player and the second player outputs the result. A protocol is a method that the players follow to compute certain functions of their input. Also the protocols can be randomized; in that case, the protocol needs to output correctly with probability at least $1 - \delta$, for some parameter $\delta \in [0, 1]$ (the probability is taken over the random coin tosses of the protocol). The randomized one-way communication complexity of a function f with error δ is denoted by $\mathcal{R}_{\delta}^{1-way}(f)$. Classically the first player is named Alice and the second player is named Bob and we also follow the same convention here. [24] is a standard reference for communication complexity.

2.4 Chernoff Bound

We will use the following concentration inequality:

Theorem 1. *Let $X_1, \ldots, X_\ell$ be a sequence of ℓ independent random variables in $[0, 1]$ (not necessarily identical). Let $S = \sum_i X_i$ and let $\mu = \mathbb{E}[S]$. Then, for any $0 \le \delta \le 1$:*

$$\Pr[|S - \mu| \ge \delta \ell] < 2 \exp(-2\ell \delta^2)$$

and

$$\Pr[|S - \mu| \ge \delta \mu] < 2 \exp(-\delta^2 \mu / 3)$$

The first inequality is called an additive bound and the second multiplicative.

2.5 Problem Definition

The basic winner determination problem is defined as follows.

Definition 1. (WINNER DETERMINATION)
Given a voting profile $\succ$ over a set of candidates $\mathcal{C}$ and a voting rule r, determine the winners $r(\succ)$.

We show a strong space complexity lower bound for the WINNER DETERMINATION problem for the plurality voting rule in Theorem 12. To overcome this theoretical bottleneck, we focus on determining *approximate winner* of an election. Below we define the notion of ε-approximate winner which we also call ε-*winner*.

Definition 2. (ε-WINNER)
Given an n-voter voting profile $\succ$ over a set of candidates $\mathcal{C}$ and a voting rule r, a candidate w is called an ε–winner if w can be made winner by changing at most εn votes in $\succ$.

Notice that there always exist an ε-winner in every election since a winner is also an ε-winner. We show that finding even an ε-winner deterministically requires large space when the number of votes is large [see Theorem 14]. However, we design space efficient randomized algorithms which outputs an ε-winner of an election with probability at least $1 - \delta$. The problem that we study here is called (ε, δ)-WINNER DETERMINATION problem and is defined as follows.

Definition 3. ((ε, δ)-WINNER DETERMINATION)
Given a voting profile $\succ$ over a set of candidates $\mathcal{C}$ and a voting rule r, determine an ε–winner with probability at least $1 - \delta$. (The probability is taken over the internal coin tosses of the algorithm.)

3 Upper Bounds

In this section, we present the algorithms for the (ε, δ)-Winner Determination problem for various voting rules. Before embarking on specific algorithms, we first prove a few supporting results that will be used crucially in our algorithms later. We begin with the following space efficient algorithm for picking an item uniformly at random from a universe of size n below.

Observation 1. *There is an algorithm for choosing an item with probability $\frac{1}{n}$ that uses $O(\log \log n)$ bits of memory and uses fair coin as its only source of randomness.*

Proof. First let us assume, for simplicity, that n is a power of 2. We toss a fair coin $\log_2 n$ many times and choose the item, say x, only if the coin comes head all the times. Hence the probability that the item x gets chosen is $\frac{1}{n}$. We need $O(\log \log n)$ space to toss the fair coin $\log_2 n$ times (to keep track of the number of times we have tossed the coin so far). If n is not a power of 2 then, toss the fair coin $\lceil \log_2 n \rceil$ many times and we choose the item x only if the coin comes head in all the tosses conditioned on some event E. The event E contains exactly n outcomes including the all heads outcome. $\qquad \square$

We remark that Observation 1 is tight in terms of space complexity. We state the claim formally below, as it may be interesting in its own right.

Proposition 1. *Any algorithm that chooses an item from a set of size n with probability p, for $0 < p \le \frac{1}{n}$, using a fair coin as its only source of randomness, must use $\Omega(\log \log n)$ bits of memory.*

Proof. The algorithm tosses the fair coin some number of times (the number of times it tosses the coin may also depend on the outcome of the previous tosses) and finally picks an item from the set. Consider a run $\mathcal{R}$ of the algorithm where it chooses the item, say x, with *smallest number of coin tosses*; say it tosses the coin t many times in this run $\mathcal{R}$. This means that in any other run of the algorithm where the item x is chosen, the algorithm must toss the coin at least t number of times. Let the outcome of the coin tosses in $\mathcal{R}$ be $r_1, \cdots, r_t$. Let s_i be the memory content of the algorithm immediately after it tosses the coin i^{th} time, for $i \in [t]$, in the run $\mathcal{R}$. First notice that if $t < \log_2 n$, then the probability with which the item x is chosen is more than $\frac{1}{n}$, which would be a contradiction. Hence, $t \ge \log_2 n$. Now we claim that all the s_i's must be different. Indeed otherwise, let us assume $s_i = s_j$ for some $i < j$. Then the algorithm chooses the item x after tossing the coin $t - (j - i)$ (which is strictly less than t) many times when the outcome of the coin tosses are $r_1, \cdots, r_i, r_{j+1}, \cdots, r_t$. This contradicts the assumption that the run $\mathcal{R}$ we started with chooses the item x with smallest number of coin tosses. $\qquad \square$

An essential ingredient in our algorithms is calculating the approximate frequencies of all the elements in a universe in an input data stream. The following result (due to [30]) provides a space efficient algorithm for that job.

Theorem 2. *Given an insertion only stream of length n over a universe of size m, there is a deterministic one pass algorithm to find the frequencies of all the items in the stream within an additive approximation of εn using $O\left(\min\{\frac{1}{\varepsilon}\left(\log m + \log n\right), m \log n\}\right)$ bits of memory, for every $\varepsilon > 0$.*

Proof. The $O\left(\frac{1}{\varepsilon}\left(\log m + \log n\right)\right)$ space algorithm is due to [30]. On the other hand, notice that with space $O\left(m \log n\right)$, we can exactly count the frequency of every element, *even in the turnstile model of stream*, by simply keeping an array of length m (indexed by ids of the elements from the universe) each entry of which is capable of storing integers up to n. $\qquad \square$

We now describe streaming algorithms for the (ε, δ)–WINNER DETERMINATION problem for various voting rules. The general idea is to sample certain number of votes uniformly at random from the stream of votes using the algorithm of Observation 1 and generate another stream of *elements* over some different universe. The number of votes sampled and the universe of the stream generated depend on the specific voting rule we are considering. After that, we approximately calculate the frequencies of the elements in the generated stream using Theorem 2. For simplicity, we assume that the number of votes in known in advance up to some constant factor (only to be able to apply Observation 1). We will see in Section 3.1 how to get rid of this assumption, without affecting space complexity of any of the algorithms much. We begin with the k-approval and k-veto voting rules below.

Theorem 3. *Assume that the number of votes is known to be within $[c_1n, c_2n]$ for some constants c_1 and c_2 in advance. Then there is a one pass algorithm for the (ε, δ)–Winner Determination problem for the k-approval voting rule that uses $O\left(\min\{\frac{k}{\varepsilon}\left(\log m + \log\frac{1}{\varepsilon} + \log\log\frac{1}{\delta}\right), m\left(\log\frac{\log(k+1)}{\varepsilon} + \log\log\frac{1}{\delta}\right)\} + \log\log n\right)$ bits of memory and for the k-veto voting rule that uses $O\left(\min\{\frac{k}{\varepsilon}\left(\log m + \log\frac{1}{\varepsilon} + \log\log\frac{1}{\delta}\right), m\left(\log\frac{\log(m-k+1)}{\varepsilon} + \log\log\frac{1}{\delta}\right)\} + \log\log n\right)$ bits of memory.*

Proof. Let us first consider the case of the k-approval voting rule. We pick the current vote in the stream with probability p (the value of p will be decided later) independent of other votes. Suppose we sample ℓ many votes; let $\mathcal{S} = \{v_i : i \in [\ell]\}$ be the set of votes sampled. From the set of sampled votes $\mathcal{S}$, we generate a stream $\mathcal{T}$ over the universe $\mathcal{C}$ as follows. For $i \in [\ell]$, let the vote v_i be $c_1 > c_2 > \cdots > c_m$. From the vote v_i, we add k candidates $c_1, \cdots, c_k$ in the stream $\mathcal{T}$. We know that there is a $\ell = O(\frac{\log(k+1)}{\varepsilon^2}\log\frac{1}{\delta})$ (and thus a corresponding $p = \Omega(\frac{1}{n})$) which ensures that for every candidate $x \in \mathcal{C}$, $|\frac{s(x)}{n} - \frac{\hat{s}(x)}{\ell}| < \frac{\varepsilon}{3}$ with probability at least $1 - \frac{\delta}{2}$ [15], where $s(\cdot)$ and $\hat{s}(\cdot)$ are the scores of the candidates in the input stream of votes and in $\mathcal{S}$ respectively. Now we count $\hat{s}(x)$ for every candidate $x \in \mathcal{C}$ within an additive approximation of $\frac{\varepsilon\ell}{3}$ and the result follows from Theorem 2 (notice that the length of the stream $\mathcal{T}$ is $k\ell$).

For the k-veto voting rule, we approximately calculate the number of vetoes that every candidate gets using the same technique as above. However, for the k-veto voting rule, the corresponding bound for ℓ is $O(\frac{\log(m-k+1)}{\varepsilon^2}\log\frac{1}{\delta})$ which implies the result. $\qquad\square$

By similar techniques, we have the following algorithm for the generalized plurality rule.

Theorem 4. *Assume that the number of votes is known to be within $[c_1n, c_2n]$ for any constants c_1 and c_2 in advance. Then there is a one pass algorithm for the (ε, δ)–Winner Determination problem for the generalized plurality voting rule that uses $O\left(\frac{1}{\varepsilon}\left(\log m + \log\frac{1}{\varepsilon} + \log\log\frac{1}{\delta}\right) + \log\log n\right)$ bits of memory.*

Proof. We sample $\ell = O(\frac{1}{\varepsilon^2}\log\frac{1}{\delta})$ many votes uniformly at random from the input stream of votes using the technique used in the proof of Theorem 3. For every candidate, we count both the number of approvals and disapprovals that it gets within an additive approximation of $\frac{\varepsilon\ell}{10}$ which is enough to get an ε-winner. Now the space complexity follows form Theorem 2. $\qquad\square$

We generalize Theorem 3 to the class of scoring rules next. We need the following result in the subsequent proof which is due to [15].

Lemma 1. *Let $\alpha = (\alpha_1, \cdots, \alpha_m)$ be an arbitrary score vector and w the winner of an α–election $\mathcal{E}$. Let x be any candidate which is not a ε–winner. Then, $s(w) - s(x) \geq \alpha_1\varepsilon n$.*

With Lemma 1 at hand, we now present the algorithm for the scoring rules.

Theorem 5. *Assume that the number of votes is known to be within $[c_1n, c_2n]$ for any constants c_1 and c_2 in advance. Let $\alpha = (\alpha_1, \cdots, \alpha_m)$ be a score vector such that $\alpha_i \geq 0$ for every $i \in [m]$. Then there is a one pass algorithm for the (ε, δ)–Winner Determination problem for the α-scoring rule that uses $O\left(\frac{\sum_{i=1}^m \alpha_i}{\alpha_1}\left(\log\log m + \log\frac{1}{\varepsilon} + \log\log\frac{1}{\delta}\right) + \log\log n\right)$, which is $O\left(m\left(\log\log m + \log\frac{1}{\varepsilon} + \log\log\frac{1}{\delta}\right) + \log\log n\right)$, bits of memory.*

Proof. Let $\alpha = (\alpha_1, \cdots, \alpha_m)$ be an arbitrary score vector with $\alpha_i \geq 0$ for every $i \in [m]$. We define $\alpha_i' = \frac{\alpha_i}{\sum_{i=j}^m \alpha_j}$ (which is in $[0,1]$), for every $i \in [m]$. Since scoring rules remain same even if we multiply every α_i with any positive constant λ, the score vectors α and α' correspond to same

voting rule. We pick the current vote in the stream with probability p (the value of p will be decided later) independent of other votes. Suppose we sample ℓ many votes; let $\mathcal{S} = \{v_i : i \in [\ell]\}$ be the set of votes sampled. For $i \in [\ell]$, let the vote v_i be $c_1 > c_2 > \cdots > c_m$. We pick the candidate c_i from the vote v_i with probability α_i' and define it to be a_i. We compute the frequencies of the candidates in the stream $\bar{\mathcal{S}} = \{a_i : i \in [\ell]\}$ within an additive factor of $\varepsilon' n$, where $\varepsilon' = \frac{\varepsilon}{3}$. For every candidate $x \in \mathcal{C}$, let $s(x)$ be the α'–score of the candidate x in the input stream of votes and $\hat{s}(x)$ be $\frac{n}{\ell}$ times the α'–score of the candidate x in the sampled votes $\mathcal{S}$. We know that there exists an $\ell = O(\frac{1}{\varepsilon^2} \log \frac{m}{\delta})$ (and thus a corresponding $p = \Omega(\frac{1}{n})$) which ensures that, for every candidate $x \in \mathcal{C}$, $|s(x) - \hat{s}(x)| < \alpha_1' \varepsilon' n$ with probability at least $1 - \frac{\delta}{2}$ [15]. Let $\bar{s}(x)$ be $\frac{n}{\ell}$ times the frequency of the candidate $x \in \mathcal{C}$ in the stream $\bar{\mathcal{S}}$. We now prove the following claim from which the result follows immediately.

Claim 1.
$$\Pr[\forall x \in \mathcal{C}, |\bar{s}(x) - \hat{s}(x)| \le \alpha_1' \varepsilon' n] \ge 1 - \frac{\delta}{2}$$

Proof. For every candidate $x \in \mathcal{C}$ and every $i \in [\ell]$, we define a random variable $X_i(x)$ to be 1 if $a_i = x$ and 0 otherwise. Then, $\bar{s}(x) = \frac{n}{\ell} \sum_{i \in [\ell]} X_i(x)$. We have, $\mathbb{E}[\bar{s}(x)] = \hat{s}(x)$. Now using Chernoff bound from Theorem 1, we have the following:

$$\Pr[|\bar{s}(x) - \hat{s}(x)| > \alpha_1' \varepsilon' n] = \Pr[|\frac{n}{\ell} \sum_{i \in [\ell]} X_i(x) - \hat{s}(x)| > \alpha_1' \varepsilon' n]$$

$$= \Pr[|\sum_{i \in [\ell]} \frac{X_i(x)}{\alpha_1'} - \frac{\ell \hat{s}(x)}{\alpha_1' n}| > \varepsilon' \ell]$$

$$\le 2 \exp\{-\frac{\varepsilon^2 \alpha_1' n \ell}{3 \hat{s}(x)}\}$$

$$\le 2 \exp\{-\frac{\varepsilon^2 \ell}{3}\}$$

The fourth inequality follows from the fact that $\hat{s}(x) \le \alpha_1' n$ for every candidate $x \in \mathcal{C}$. Now we use the union bound to get the following.

$$\Pr[\forall x \in \mathcal{C}, |\bar{s}(x) - \hat{s}(x)| \le \alpha_1' \varepsilon' n] \ge 1 - \sum_{x \in \mathcal{C}} 2 \exp\{-\frac{\varepsilon^2 \ell}{3}\} \ge 1 - \frac{\delta}{2}$$

The second inequality follows from an appropriate choice of $\ell = O(\frac{1}{\varepsilon^2} \log \frac{m}{\delta})$. □

We estimate the frequency of every candidate in $\bar{\mathcal{S}}$ within an additive approximation ratio of $\alpha_1' \varepsilon \ell$ and output the candidate w with maximum estimated frequency as the winner of the election. The candidate w is an $\varepsilon-$ winner (follows from Lemma 1) with probability at least $1 - \delta$ (follows from Claim 1). The space complexity of this algorithm follows from Theorem 2 (since $\frac{1}{\alpha_1'} = \frac{\sum_{i=1}^m \alpha_i}{\alpha_1} \le \frac{m \alpha_1}{\alpha_1} = m$) and Observation 1. □

We present next the streaming algorithm for the approval voting rule. It is again obtained by running a frequency estimation algorithm on samples from a stream.

Theorem 6. *Assume that the number of votes is known to be within $[c_1 n, c_2 n]$ in advance, for some constants c_1 and c_2. Then there is a one pass algorithm for the (ε, δ)–WINNER DETERMINATION problem for the approval voting rule that uses $O\left(m\left(\log\log m + \log \frac{1}{\varepsilon} + \log\log \frac{1}{\delta}\right) + \log\log n\right)$ bits of memory.*

Proof. We sample ℓ many votes using the algorithm described in Observation 1 and technique described in the proof of Theorem 5. The total number of approvals in those sampled votes is at most $m\ell$ and we estimate the number of approvals that every candidate receives within an additive approximation of $\frac{\varepsilon\ell}{2}$. The result now follows from the upper bound on ℓ [15] and Theorem 2. □

Now we move on to maximin, Copeland, Bucklin, and plurality with run off voting rules. We provide two algorithms for these voting rules, which trade off between the number of candidates m and the approximation factor ε. The algorithm in Theorem 7 below, which has better space complexity when $\frac{1}{\varepsilon}$ is small compared to m, simply stores all the sampled votes.

Theorem 7. *Assume that the number of votes is known to be within $[c_1 n, c_2 n]$ in advance, for some constants c_1 and c_2. Then there is a one pass algorithm for the (ε, δ)–WINNER DETERMINATION problem for the maximin, Bucklin, and plurality with run off voting rules that use $O\left(\frac{m \log^2 m \log \frac{1}{\delta}}{\varepsilon^2} + \log \log n\right)$ bits of memory and for the Copeland voting rule that uses $O\left(\frac{m \log^4 m \log \frac{1}{\delta}}{\varepsilon^2} + \log \log n\right)$ bits of memory.*

Proof. We sample ℓ many votes from the input stream of votes uniformly at random and simply store all of them. Notice that we can store a vote using space $O(m \log m)$. The result now follows from the upper bound on ℓ [15] and Observation 1. □

Next we consider the case when $\frac{1}{\varepsilon}$ is large compared to m.

Theorem 8. *Assume that the number of votes is known to be within $[c_1 n, c_2 n]$ in advance, for some constants c_1 and c_2. Then there is a one pass algorithm for the (ε, δ)–WINNER DETERMINATION problem for the maximin, Copeland, Bucklin, and plurality with runoff voting rules that uses $O\left(m^2 \left(\log \log m + \log \frac{1}{\varepsilon} + \log \log \frac{1}{\delta}\right) + \log \log n\right)$ bits of memory.*

Proof. For each voting rule mentioned in the statement, we sample ℓ many votes $\mathcal{S} = \{v_i : i \in [\ell]\}$ uniformly at random from the input stream of votes using the algorithm used in Observation 1 and the technique used in the proof of Theorem 5. From $\mathcal{S}$, we generate another stream $\bar{\mathcal{S}}$ of elements belonging to a different universe $\mathcal{U}$ (which depends on the voting rule under consideration). Finally, we calculate the frequencies of the elements of $\bar{\mathcal{S}}$, using Theorem 2, within an additive approximation of $\frac{\varepsilon\ell}{2}$ for maximin, Bucklin, and plurality with runoff voting rules and $\frac{\varepsilon\ell}{2 \log m}$ for the Copeland voting rule. The difference of approximation factor is due to [15]. We know that $\ell = O\left(\frac{\log \frac{m}{\delta}}{\varepsilon^2}\right)$ for maximin, Bucklin, and plurality with run off voting rules and $\ell = O\left(\frac{\log^3 \frac{m}{\delta}}{\varepsilon^2}\right)$ for the Copeland voting rule [15]. This bounds on ℓ prove the result once we describe $\bar{\mathcal{S}}$ and $\mathcal{U}$. Below, we describe the stream $\bar{\mathcal{S}}$ and the universe $\mathcal{U}$ for individual voting rules. Let the vote v_i be $c_1 \succ c_2 \succ \cdots \succ c_m$.

- **maximin, Copeland:** $\mathcal{U} = \mathcal{C} \times \mathcal{C}$. From the vote v_i, we put (c_j, c_k) in $\bar{\mathcal{S}}$ for every $j < k$.

- **Bucklin:** $\mathcal{U} = \mathcal{C} \times [m]$. From the vote v_i, we put (c_j, k) in $\bar{\mathcal{S}}$ for every $j \leq k$.

- **plurality with runoff:** $\mathcal{U} = \mathcal{C} \times \mathcal{C}$. From the vote v_i, we put (c_j, c_k) in $\bar{\mathcal{S}}$ for every $j < k$ and (c_1, c_1). In the plurality with runoff voting rule, we need to estimate the plurality score of every candidate which we do by estimating the frequencies of the elements of the (x, x) in $\bar{\mathcal{S}}$. We also need to estimate $D_{\mathcal{E}}(x, y)$ for every candidate $x, y \in \mathcal{C}$ which we do by estimating the frequencies of the elements of the form (x, y). □

11

3.1 Unknown stream length

Now we consider the case when the number of voters is not known beforehand. The idea is to use reservoir sampling ([35]) along with approximate counting ([18, 31]) to pick an element from the stream *almost uniformly* at random. The following result shows that we can do so in a space efficient manner.

Theorem 9. *(Theorem 7 of [19]) Given an insertion only stream of length n (n is not known to the algorithm beforehand) over a universe of size m, there is a randomized one pass algorithm that outputs, with probability at least $1 - \delta$, the element at a random position $X \in [n]$ such that, for every $i \in [n], |\Pr\{X = i\} - \frac{1}{n}| \le \frac{\varepsilon}{n}$ using $O(\log \frac{1}{\delta} + \log \frac{1}{\varepsilon} + \log \log n + \log m)$ bits of memory, for every $\varepsilon \in (0, 1]$ and $\delta > 0$.*

Recall that Theorem 2 only works for insertion only streams. However, as the stream progresses, the element chosen by Theorem 9 changes; so, we cannot invoke Misra-Gries to do frequency estimation on a set of samples given by Theorem 9. For streams with both insertions and deletions, we have the following result which is due to count-min sketch [12].

Theorem 10. *Given a turnstile stream of length n over a universe of size m, there is a randomized one pass algorithm to find the frequencies of the items in the stream within an additive approximation of εn with probability at least $1 - \delta$ using $O\left(\frac{\log m}{\varepsilon} \log(\frac{1}{\delta})(\log m + \log n)\right)$ bits of memory, for every $\varepsilon > 0$ and $\delta > 0$.*

From Theorem 9 and 10 and from the proofs of Theorem 2, 4 to 6 and 8, we get the following.

Corollary 1. *Assume that the number of votes n is not known beforehand. Then there is a one pass algorithm for the (ε, δ)–WINNER DETERMINATION problem for k-approval, k-veto, generalized plurality, approval, maximin, Copeland, Bucklin, and plurality with run off voting rules that uses $\log m \log \frac{1}{\delta}$ times more space than the corresponding algorithms when n is known beforehand upto a constant factor.*

Proof. We use reservoir sampling with approximate counting from Theorem 9. The resulting stream that we generate have both positive and negative updates (since in reservoir sampling, we sometimes replace an item we previously sampled). Now we approximately estimate the frequency of every item in the generated stream using Theorem 10. □

Again from Theorem 7 and 9, we get the following result which provides a better space upper bound than Corollary 1 when the number of candidates m is large.

Corollary 2. *Assume that the number of votes n is not known beforehand. Then there is a one pass algorithm for the (ε, δ)–WINNER DETERMINATION problem for the maximin, Bucklin, and plurality with run off voting rules that use $O\left(\frac{m \log^2 m \log \frac{1}{\delta}}{\varepsilon^2} + \log \log n\right)$ bits of memory and for the Copeland voting rule that uses $O\left(\frac{m \log^4 m \log \frac{1}{\delta}}{\varepsilon^2} + \log \log n\right)$ bits of memory.*

3.2 Sliding Window Model

Suppose we want to compute an ε-winner of the last n many votes in an infinite stream of votes for various voting rules. The following result shows that there is an algorithm, with space complexity same as Theorem 9, to sample a vote from the last n votes in a stream.

Theorem 11. *([6]) Given an insertion only stream over a universe of size m, there is a randomized one pass algorithm that outputs, with probability at least $1 - \delta$, the element at a random position X from last n positions such that, for every $i \in [n], |\Pr\{X = i\} - \frac{1}{n}| \leq \frac{\varepsilon}{n}$ using $O(\log \frac{1}{\delta} + \log \frac{1}{\varepsilon} + \log \log n + \log m)$ bits of memory, for every $\varepsilon \in (0, 1]$ and $\delta > 0$.*

Theorem 11 immediately provides results same as Corollary 1 and 2, where n is the window size.

4 Lower Bounds

In this section, we prove space complexity lower bounds for the (ε, δ)–WINNER DETERMINA-TION problem for various voting rules. We reduce certain communication problems to the (ε, δ)–WINNER DETERMINATION problem for proving space complexity lower bounds. Let us first introduce those communication problems with necessary results.

4.1 Communication Complexity

Definition 4. *(*AUGMENTED-INDEXING$_{m,t}$*)*
Let t and m be positive integers. Alice is given a string $x = (x_1, \cdots, x_t) \in [m]^t$. Bob is given an integer $i \in [t]$ and $(x_1, \cdots, x_{i-1})$. Bob has to output x_i.

The following communication complexity lower bound result is due to [16] by a simple extension of the arguments of Bar-Yossef et al [4].

Lemma 2. $\mathcal{R}_\delta^{1-way}($AUGMENTED-INDEXING$_{m,t}) = \Omega((1 - \delta)t \log m)$ *for any $\delta < 1 - \frac{3}{2m}$.*

Also, we recall the multi-party version of the set-disjointness problem.

Definition 5. *(*DISJ$_{m,t}^{promise}$*)*
We have t sets $X_1, \cdots, X_t$ each a subset of $[m]$. We have t players and player i is holding the set X_i. We are also given the promise that either $X_i \cap X_j = \varnothing$ for every $i \neq j$ or there exist an element $y \in [m]$ such that $y \in X_i$ for every $i \in [t]$ and $(X_i \setminus \{y\}) \cap (X_j \setminus \{y\}) = \varnothing$ for every $i \neq j$. The output DISJ$_{m,t}^{promise}(X_1, \cdots, X_t)$ *is 1 if $X_i \cap X_j = \varnothing$ for every $i \neq j$ and 0 else.*

Lemma 3 (Proved in [4, 8].). $\mathcal{R}_\delta^{1-way}($DISJ$_{m,t}^{promise}) = \Omega(\frac{m}{t})$, *for any $\delta \in [0, 1)$ and t.*

The following communication problem is very useful for us.

Definition 6. *(*MAX-SUM$_{m,t}$*)*
Alice is given a string $x = (x_1, x_2, \cdots, x_t) \in [m]^t$ of length t over universe $[m]$. Bob is given another string $y = (y_1, y_2, \cdots, y_t) \in [m]^t$ of length t over the same universe $[m]$. The strings x and y is such that the index i that maximizes $x_i + y_i$ is unique. Bob has to output the index $i \in [t]$ which satisfies $x_i + y_i = \max_{j \in [t]}\{x_j + y_j\}$.

We establish the following one way communication complexity lower bound for the MAX-SUM$_{m,t}$ problem by reducing it from the AUGMENTED-INDEXING$_{2,t \log m}$ problem.

Lemma 4. $\mathcal{R}_\delta^{1-way}($MAX-SUM$_{m,t}) = \Omega(t \log m)$, *for every $\delta < \frac{1}{4}$.*

Proof. We reduce the AUGMENTED-INDEXING$_{2,t\log m}$ problem to MAX-SUM$_{8m,t+1}$ problem thereby proving the result. Let the inputs to Alice and Bob in the AUGMENTED-INDEXING$_{2,t\log m}$ instance be $(a_1, a_2, \cdots, a_{t\log m}) \in \{0,1\}^{t\log m}$ and $(a_1, \cdots, a_{i-1})$ respectively. The idea is to construct a corresponding instance of the MAX-SUM$_{8m,t+1}$ problem that outputs $t+1$ if and only if $a_i = 0$. We achieve this as follows. Alice starts execution of the MAX-SUM$_{8m,t+1}$ protocol using the vector $x = (x_1, x_2, \cdots, x_{t+1}) \in [8m]^{t+1}$ which is defined as follows: the binary representation of x_j is $\left(0, 0, a_{(j-1)\log m+1}, a_{(j-1)\log m+2}, a_{(j-1)\log m+3}, \cdots, a_{j\log m}, 0\right)_2$, for every $j \in [t]$, and x_{t+1} is 0. Bob participates in the MAX-SUM$_{8m,t+1}$ protocol with the vector $y = (y_1, y_2, \cdots, y_{t+1}) \in [8m]^{t+1}$ which is defined as follows. Let us define $\lambda = \left\lceil \frac{i}{\log m} \right\rceil$. We define $y_j = 0$, for every $j \notin \{\lambda, t+1\}$. The binary representation of y_λ is $(1, 0, a_{(\lambda-1)\log m+1}, a_{(\lambda-1)\log m+2}, \cdots, a_{i-1}, 1, 0, 0, \cdots, 0, 0, 1)_2$. Let us define an integer T whose binary representation is $(0, 0, a_{(\lambda-1)\log m+1}, a_{(\lambda-1)\log m+2}, \cdots, a_{i-1}, 0, 1, 1, \cdots, 1)_2$. We define y_{t+1} to be $T + y_\lambda$. First notice that the output of the MAX-SUM$_{8m,t+1}$ instance is either λ or $t+1$, by the construction of y. Now observe that if $a_i = 1$ then, $x_\lambda > T$ and thus the output of the MAX-SUM$_{8m,t+1}$ instance should be λ. On the other hand, if $a_i = 0$ then, $x_\lambda < T$ and thus the output of the MAX-SUM$_{8m,t+1}$ instance should be $t+1$. □

Finally, we also consider the GREATER-THAN problem.

Definition 7. *(*GREATER-THAN$_n$*)*
Alice is given an integer $x \in [n]$ and Bob is given an integer $y \in [n], y \neq x$. Bob has to output 1 if $x > y$ and 0 otherwise.

The following result is due to [29, 34]. We provide a simple proof of it that seems to be missing[IV] in the literature.

Lemma 5. $\mathcal{R}_\delta^{1-way}($GREATER-THAN$_n) = \Omega(\log n)$, *for every $\delta < \frac{1}{4}$.*

Proof. We reduce the AUGMENTED-INDEXING$_{2,\lceil \log n \rceil+1}$ problem to the GREATER-THAN$_n$ problem thereby proving the result. Alice runs the GREATER-THAN$_n$ protocol with its input number whose representation in binary is $a = (x_1 x_2 \cdots x_{\lceil \log n \rceil} 1)_2$. Bob participates in the GREATER-THAN$_n$ protocol with its input number whose representation in binary is $b = (x_1 x_2 \cdots x_{i-1} 1 \underbrace{0 \cdots 0}_{(\lceil \log n \rceil - i + 1)\ 0's})_2$.

Now $x_i = 1$ if and only if $a > b$. □

4.2 Reductions

4.2.1 The cases $\varepsilon = 0$ and $\delta = 0$

We begin with the problem where we have to find the winner (i.e., 0-winner) for a plurality election. Notice that, we can find the winner by exactly computing the plurality score of every candidate. This requires $O(m \log n)$ bits of memory. We prove below that, when n is much larger than m, this space complexity is *almost* optimal even if we are allowed to use randomization, by reducing it from the MAX-SUM$_{n,m}$ problem. This strengthens a similar result proved in Karp et al. [22] only for deterministic algorithms.

Theorem 12. *Any one pass $(0,\delta)$–WINNER DETERMINATION algorithm for the plurality and generalized plurality election must use $\Omega(m \log(n/m))$ bits of memory, for any $\delta \in [0, \frac{1}{4})$.*

[IV]A similar proof appears in [23] but theirs gives a weaker lower bound.

Proof. We prove the result for $(0,\delta)$–WINNER DETERMINATION problem for the plurality election. This gives the result for the generalized plurality election since every plurality election is also a generalized plurality election. Consider the MAX-SUM$_{n,m}$ problem where Alice is given a string $x = (x_1, \cdots, x_m) \in [n]^m$ and Bob is given another string $y = (y_1, \cdots, y_m) \in [n]^m$. The candidate set of our election is $[m]$. The votes would be such that the only winner will be the candidate i such that $i \in \arg\max_{j \in [m]} \{x_j + y_j\}$. Moreover, the winner would be known to Bob, thereby proving the result. Thus Bob can output x_i correctly whenever our $(0,\delta)$–WINNER DETERMINATION algorithm outputs correctly. Alice generates x_j many plurality votes for the candidate j, for every $j \in [m]$. Alice now sends the memory content to Bob. Bob resumes the run of the algorithm by generating y_j many plurality votes for the candidate j, for every $j \in [m]$. The plurality score of candidate j is $(x_j + y_j)$ and thus the plurality winner will be a candidate i such that $i \in \arg\max_{j \in [m]} \{x_j + y_j\}$. Notice that the total number of votes is at most $2mn$. The result now follows from Lemma 4. $\square$

For the case when m and n are comparable, the following result is stronger. We prove this by exhibiting a reduction from the DISJ$_{m,3}^{promise}$ problem.

Theorem 13. *Any one pass* $(0,\delta)$–WINNER DETERMINATION *algorithm for the plurality and generalized plurality election must use* $\Omega(\min\{m,n\})$ *bits of memory, for any* $\delta \in [0,1)$.

Proof. Suppose we have a one pass $(0,\delta)$–WINNER DETERMINATION algorithm for the plurality election that uses s bits of memory. We will demonstrate a one-way three party protocol to compute DISJ$_{m,3}^{promise}$ function using $2s$ bits of communication thus proving the result. We have the candidate set $[m+1]$. The protocol is as follows.

Player 1 starts running the one pass $(0,\delta)$–WINNER DETERMINATION algorithm on the input $X_1 \cup \{m+1\}$. Once player 1 is done reading all its input, it sends its memory content to player 2. This needs at most s bits of communication. Player 2 resumes the run of the algorithm with input $X_2 \cup \{m+1\}$ and sends its memory content to player 3. Again this needs at most s bits of communication. Player 3 resumes the run of the algorithm on input X_3 and output 1 if and only if the winner is $m+1$ and 0 else. Notice that, if the $X_i \cap X_j = \varnothing$ for every $i \neq j$ then, the only winner of the votes $(X_1, m+1, X_2, m+1, X_3)$ is the candidate $m+1$ with a plurality score of two. On the other hand, if there exist an element $y \in [m]$ such that $y \in X_i$ for every $i \in [t]$ and $(X_i \setminus \{y\}) \cap (X_j \setminus \{y\}) = \varnothing$ for every $i \neq j$ then, the only winner of the votes $(X_1, m+1, X_2, m+1, X_3)$ is the candidate y with a plurality score of three.

The number of candidates in the election above is $m+1$ and the number of votes n is $|X_1| + |X_2| + |X_3| + 2(m+1) = \Theta(m)$. This gives a space complexity lower bound of $\Omega(\min\{m,n\})$. $\square$

Theorem 12 and 13 give space complexity lower bounds for the case $\varepsilon = 0$. Next, we consider the other extreme case: deterministically find an ε-winner, corresponding to $\delta = 0$.

Theorem 14. *Assume* $\varepsilon < \frac{1}{5}$. *Then any one pass* $(\varepsilon, 0)$–WINNER DETERMINATION *algorithm for the plurality election must use* $\Omega(\log n)$ *bits of memory, even if the number of voters is known up to a factor of* 2 *and the number of candidates is only* 2. *The same applies for generalized plurality, scoring rules, maximin, Copeland, Bucklin, and plurality with run off voting rules.*

Proof. For the sake of contradiction, we assume that the number of possible memory contents of the algorithm is $o(n)$, since otherwise the algorithm uses $\Omega(\log n)$ space and we have nothing to prove. Our candidate set is $\{0,1\}$. We will generate two vote streams, say R_1 and R_2, in such a way that the final state of the algorithm would be same; however ε–winner would be different for the two streams thus providing the contradiction we are looking for.

Let s_0 be the starting state of the algorithm. Consider the stream of votes for 1 and let the algorithm repeats its state for the first time after reading i many 1 votes. Let the state of the algorithm after reading i^{th} 1 vote be same as the state the algorithm was after it read j^{th} 1 vote. Let us call $\mu = i - j$. Clearly $\mu = o(n)$. Then there exist $\delta_1, \delta_2 = o(n)$ such that the state the algorithm will be after reading $\frac{n}{4} - \delta_1$ many votes for 1 is same as the state it will be after reading $\frac{3n}{4} + \delta_2$ many votes for 1. Let R_1 be the stream of $\frac{n}{4} - \delta_1$ many votes for 1 followed by $\frac{n}{2}$ many votes for 0. Let R_2 be the stream of $\frac{3n}{4} + \delta_2$ many votes for 1 followed by $\frac{n}{2}$ many votes for 0. By construction the output of the algorithm is same for both the streams R_1 and R_2. However, candidate 1 is only ε-winner in R_1 and candidate 0 is only ε-winner in R_2.

For elections with two candidates, scoring rules, maximin, Copeland, Bucklin, and plurality with run off voting rules are same as the plurality voting rule. $\qquad\square$

4.2.2 Lower Bounds for Approximate and Randomized algorithms

Now we move on and show space complexity lower bounds for general (ε, δ)–WINNER DETERMINATION problem for various voting rules. The observation below immediately follows from the fact that the algorithm has to output a candidate as an ε-winner.

Observation 2. *Every (ε, δ)–WINNER DETERMINATION algorithm, for all the voting rules considered in this paper, needs $\Omega(\log m)$ bits of memory.*

We show next a space complexity lower bound of $\Omega(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ bits for the (ε, δ)–WINNER DETERMINATION problem for various voting rules.

Theorem 15. *Suppose the number of candidates m is at least $\frac{1}{\varepsilon}$. Any one pass (ε, δ)–WINNER DETERMINATION algorithm for approval, k-approval, for $k = O(m^\lambda)$ for every $\lambda \in [0, 1)$, generalized plurality, Borda, maximin, Copeland, and plurality with run off elections must use $\Omega((1-\delta)\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ bits of memory, even when the number of votes are exactly known beforehand, for every $1 - \delta > \frac{3\varepsilon}{2}$.*

Proof. We will show that, when $m \geq \frac{1}{\varepsilon}$, we need $\Omega(\frac{1}{\sqrt{\varepsilon}} \log \frac{1}{\varepsilon})$ bits of memory for solving the $(\frac{\sqrt{\varepsilon}}{8}, \delta)$–WINNER DETERMINATION problem, thereby proving the result. Consider the AUGMENTED-INDEXING$_{1/\sqrt{\varepsilon}, 1/\sqrt{\varepsilon}}$ problem where Alice is given a string $x = (x_1, x_2, \cdots, x_{1/\sqrt{\varepsilon}}) \in [1/\sqrt{\varepsilon}]^{1/\sqrt{\varepsilon}}$ and Bob is given an integer $i \in [1/\sqrt{\varepsilon}]$ and $(x_1, \cdots, x_{i-1})$. The candidate set of the election, that we generate, is $[1/\sqrt{\varepsilon}] \times [1/\sqrt{\varepsilon}]$. The overview of the technique is as follows: Alice generates a stream of votes and runs the algorithm, then sends the memory content to Bob, and Bob resumes the run of the algorithm with another stream of votes (both the streams of votes depend on the voting rule under consideration) in such a way that the only $\frac{\sqrt{\varepsilon}}{8}$–winner will be the candidate (x_i, i). Thus Bob can output x_i correctly if and only if the $(\sqrt{\varepsilon}/8, \delta)$–WINNER DETERMINATION algorithm outputs correctly. Now the result follows from Lemma 2. The elections for specific voting rules are as follows. Let n be the number of votes.

- **k-approval for $k = O(m^\lambda)$ for every $\lambda \in [0, 1)$, approval, and generalized plurality:** It is enough to prove the result for the k-approval voting rule for $k = O(m^\lambda)$ for every $\lambda \in [0, 1)$, since every k-approval election is also an approval election. For $k = 1$, we get the result for the plurality voting rule and thus for the generalized plurality voting rule, since every plurality election is also a generalized plurality election.

  - **Case 1: $k \leq \sqrt{m}$:** Alice generates a stream of $\frac{n}{2}$ votes in such a way that the k-approval score of every candidate in $\{(x_j, j) : j \in [1/\sqrt{\varepsilon}]\}$ is at least $\lfloor k\sqrt{\varepsilon}n/2 \rfloor$ and the k-approval score of any other candidate is 0. Alice now sends the memory content of the algorithm to Bob. Bob resumes the run of the algorithm by generating another stream of $n/2$ votes

16

in such a way that the k-approval score of every candidate in $\{(j,i) : j \in [1/\sqrt{\varepsilon}]\}$ is at least $\lfloor k\sqrt{\varepsilon}n/2 \rfloor$ and the k-approval score of any other candidate is 0. The score of the candidate (x_i, i) is at least $\lfloor k\sqrt{\varepsilon}n \rfloor$ where as the score of every other candidate is at most $\lceil k\sqrt{\varepsilon}n/2 \rceil$. Hence the only $\sqrt{\varepsilon}/8$–winner is (x_i, i).

- **Case 2:** $k > \sqrt{m}$ and $k = O(m^\lambda)$ for any $\lambda \in [0.5, 1)$: Alice generates a stream of $\frac{n}{2}$ votes in such a way that the k-approval score of every candidate in $\{(x_j, j) : j \in [\frac{1}{\sqrt{\varepsilon}}]\}$ is at least $\frac{n}{2}$ and the k-approval score of any other candidate is at most $\lceil (k - \frac{1}{\sqrt{\varepsilon}})n/\left(\frac{2}{\sqrt{\varepsilon}}(\frac{1}{\sqrt{\varepsilon}}-1)\right)\rceil$, which is at most $\frac{n}{2} - \frac{\sqrt{\varepsilon}}{2}n$ for sufficiently small constant ε (depending on λ). Alice now sends the memory content of the algorithm to Bob. Bob resumes the run of the algorithm by generating another stream of $\frac{n}{2}$ votes in such a way that the k-approval score of every candidate in $\{(j,i) : j \in [\frac{1}{\sqrt{\varepsilon}}]\}$ is at least $\frac{n}{2}$ and the k-approval score of any other candidate is $\lceil (k - \frac{1}{\sqrt{\varepsilon}})\frac{n}{\frac{2}{\sqrt{\varepsilon}}(\frac{1}{\sqrt{\varepsilon}}-1)}\rceil$. In this case also the only $\frac{\sqrt{\varepsilon}}{8}$–winner is (x_i, i).

- **Borda, Bucklin:** Alice generates a stream of $\frac{n}{2}$ votes where the candidates in $\{(x_\ell, \ell), \ell \in [1/\sqrt{\varepsilon}]\}$ are uniformly distributed in top $1/\sqrt{\varepsilon}$ positions of the votes and the rest of the candidates are uniformly distributed in bottom $1/\varepsilon - 1/\sqrt{\varepsilon}$ positions of the votes. Alice now sends the memory content to Bob and Bob resumes the run of the algorithm by generating another stream of $n/2$ votes where the candidates in $\{(\ell, i), \ell \in [1/\sqrt{\varepsilon}]\}$ are uniformly distributed in top $1/\sqrt{\varepsilon}$ positions of the votes and the rest of the candidates are uniformly distributed in bottom $1/\varepsilon - 1/\sqrt{\varepsilon}$ positions of the votes. The Borda score of the candidate (x_i, i) is $(1/\varepsilon - 1/2\sqrt{\varepsilon})n$ whereas the Borda score of every other candidate is at most $(1/2\varepsilon - 1/4\sqrt{\varepsilon})n$. Hence, the only $\sqrt{\varepsilon}/8$–winner for the Borda voting rule is (x_i, i), since each vote change can reduce or increase the Borda score of any candidate by at most $1/\varepsilon$.

  The candidate (x_i, i) is ranked within top $2/3\sqrt{\varepsilon}$ positions in $2n/3$ many votes, whereas any other candidate is ranked within top $2/3\sqrt{\varepsilon}$ positions in at most $n/3$ many votes. Hence the only $\sqrt{\varepsilon}/8$–winner for the Bucklin voting rule is (x_i, i).

- **Any Condorcet consistent voting rule, Plurality with runoff:** Let us define $X = \{(x_\ell, \ell) : \ell \in [1/\sqrt{\varepsilon}]\}$, $Y = [1/\sqrt{\varepsilon}] \times [1/\sqrt{\varepsilon}] \setminus X$. Suppose $\overrightarrow{X}$ and $\overrightarrow{Y}$ are arbitrary but fixed ordering of the candidates in X and Y respectively. For every $\ell \in [1/\sqrt{\varepsilon}]$, Alice generates $\sqrt{\varepsilon}n/4$ votes of the form $(x_\ell, \ell) \succ \overrightarrow{X \setminus \{(x_\ell, \ell)\}} \succ \overrightarrow{Y}$ and another $\sqrt{\varepsilon}n/4$ votes of the form $\overleftarrow{X \setminus (x_\ell, \ell)} \succ (x_\ell, \ell) \succ \overrightarrow{Y}$, where $\overleftarrow{X}$ is the reverse order of $\overrightarrow{X}$. Alice now sends the memory content to Bob. Let us define $A = \{(\ell, i) : \ell \in [1/\sqrt{\varepsilon}]\}$ and $B = [1/\sqrt{\varepsilon}] \times [1/\sqrt{\varepsilon}] \setminus A$. Suppose $\overrightarrow{A}$ and $\overrightarrow{B}$ are arbitrary but fixed ordering of A and B respectively. Bob resumes the run of the algorithm by generating another $\sqrt{\varepsilon}n/4$ votes of the form $(\ell, i) \succ \overrightarrow{A \setminus (\ell, i)} \succ \overrightarrow{B}$ and another $\sqrt{\varepsilon}n/4$ votes of the form $\overleftarrow{A \setminus (\ell, i)} \succ (\ell, i) \succ \overrightarrow{B}$ for every $\ell \in [1/\sqrt{\varepsilon}]$, where $\overleftarrow{A}$ is the reverse order of $\overrightarrow{A}$. The candidate (x_i, i) defeats every other candidate in pairwise election by a margin of at least $\frac{n}{4}$. Also the plurality score of the candidate (x_i, i) is more than the plurality score of every other candidate by at least $\sqrt{\varepsilon}n$. Hence the only $\sqrt{\varepsilon}/8$–winner is (x_i, i). $\qquad\square$

We can prove a space lower bound of $\Omega(m\varepsilon \log \frac{1}{\varepsilon})$ for one pass (ε, δ)–WINNER DETERMINATION algorithms for Borda, Bucklin, Copeland, and maximin voting rules by reducing it from AUGMENTED-INDEXING$_{1/\varepsilon, m}$ in the proof of Theorem 15. We summarize this observation below.

Corollary 3. *Suppose the number of candidates m is at least $\frac{1}{\varepsilon}$. Any one pass (ε, δ)–WINNER DETERMINATION algorithm for Borda, maximin, Copeland, and plurality with run off elections must use $\Omega((1 - \delta)m \log \frac{1}{\varepsilon})$ bits of memory, even when the number of votes are exactly known beforehand, for every $1 - \delta > \frac{3\varepsilon}{2}$.*

For the k-veto voting rule, we prove below, again by reducing from Augmented-indexing, a slightly weaker space complexity lower bound compared to the bounds of Theorem 15.

Theorem 16. *Suppose the number of candidates m is at least $\frac{1}{\varepsilon}$. Any one pass (ε, δ)–Winner Determination algorithm for the k-veto voting rule for $k = O(m^\lambda)$, for every $\lambda \in [0, 1)$, must use $\Omega(\frac{1}{\varepsilon^\mu} \log \frac{1}{\varepsilon})$, for every constant $\mu < 1$, bits of memory, even when the number of votes are exactly known beforehand, for every $1 - \delta > \frac{3\varepsilon}{2}$.*

Proof. We prove the result for $(\frac{\varepsilon}{5}, \delta)$–Winner Determination problem. Consider the Augmented-indexing$_{\frac{1}{\varepsilon^{1-\mu}}, \frac{1}{\varepsilon^\mu}}$ problem where the first player Alice is given a string $x \in [\frac{1}{\varepsilon^{1-\mu}}]^{\frac{1}{\varepsilon^\mu}}$, while the second player Bob is given an integer $i \in [\frac{1}{\varepsilon^\mu}]$ and x_j for every $j < i$. The candidate set of our election is $[\frac{1}{\varepsilon^{1-\mu}}] \times [\frac{1}{\varepsilon^\mu}]$. The votes would be such that the only $\frac{\varepsilon}{5}$–winner will be the candidate (x_i, i), thereby proving the result. Thus Bob can output x_i correctly whenever our (ε, δ)–Winner Determination algorithm outputs correctly. Alice generates a stream of $\frac{n}{2}$ votes (assume n to be sufficiently large) in such a way that for every $a, b \in \{(x_j, j) : j \in \frac{1}{\varepsilon^\mu}\}$ and $x, y \in [\frac{1}{\varepsilon^{1-\mu}}] \times [\frac{1}{\varepsilon^\mu}] \setminus \{(x_j, j) : j \in \frac{1}{\varepsilon^\mu}\}$, we have $s(a) - s(x) \geq \frac{\varepsilon n}{2}$, $s(b) - 1 \leq s(a) \leq s(b) + 1$, and $s(y) - 1 \leq s(x) \leq s(y) + 1$, where $s(\cdot)$ is the number of vetoes that a candidate receives (which is always negative or zero). This is possible since $k = O(m^\lambda)$ for $\lambda \in [0, 1)$. Alice now sends the memory content of the algorithm. Bob resumes the run of the algorithm by generating another stream of $\frac{n}{2}$ votes in such a way that for every $a', b' \in \{(z, i) : z \in \frac{1}{\varepsilon^{1-\mu}}\}$ and $x', y' \in [\frac{1}{\varepsilon^{1-\mu}}] \times [\frac{1}{\varepsilon^\mu}] \setminus \{(z, i) : z \in \frac{1}{\varepsilon^{1-\mu}}\}$, we have $s(a') - s(x') \geq \frac{\varepsilon n}{2}$, $s(b') - 1 \leq s(a') \leq s(b') + 1$, and $s(y') - 1 \leq s(x') \leq s(y') + 1$. Now the score of (x_i, i) is more than the score of every other candidate by at least $\frac{\varepsilon n}{2}$. Hence, the candidate (x_i, i) is the unique $\frac{\varepsilon}{5}$–winner. $\qquad\square$

For the k-approval voting rule, we provide a stronger space complexity lower bound of $\Omega(\frac{k}{\varepsilon} \log \frac{1}{\varepsilon})$, when the number of candidates m is at least $\frac{k}{\varepsilon^2}$, by reducing from Augmented-indexing$_{\frac{1}{\varepsilon}, \frac{k}{\varepsilon}}$.

Theorem 17. *Assume that the number of candidates m is at least $\frac{k}{\varepsilon^2}$. Then any one pass (ε, δ)–Winner Determination algorithm for the k-approval voting rule must use $\Omega(\frac{k}{\varepsilon} \log \frac{1}{\varepsilon})$ bits of memory.*

Proof. We prove the result for $(\frac{\varepsilon}{5}, \delta)$–Winner Determination problem. Consider the Augmented-indexing$_{\frac{1}{\varepsilon}, \frac{k}{\varepsilon}}$ problem where Alice is given $(x_1, \cdots, x_{\frac{k}{\varepsilon}}) \in [\frac{1}{\varepsilon}]^{\frac{k}{\varepsilon}}$ and Bob is given $(x_1, \cdots, x_{i-1})$. We will create a k-approval election in such a way that the $\frac{\varepsilon}{5}$-winner will reveal x_i to Bob. The candidate set of our election is $[\frac{1}{\varepsilon}] \times [\frac{k}{\varepsilon}]$. For every $j \in [k]$, Alice generates $\frac{\varepsilon n}{2}$ many votes approving candidates in $\{(x_{k(j-1)+1}, k(j-1)+1), (x_{k(j-1)+2}, k(j-1)+2), \cdots, (x_{kj}, kj)\}$. Alice now sends the memory content to Bob. Let $\mathcal{X} = \{(j, i) : j \in [\frac{1}{\varepsilon}]\}$. If $k \leq \frac{1}{\varepsilon}$ then, Bob generates $\frac{n}{2}$ votes in such a way that every candidate in $\mathcal{X}$ gets at least $\frac{k\varepsilon n}{2}$ many approvals and the candidates in $[\frac{1}{\varepsilon}] \times [\frac{k}{\varepsilon}] \setminus \mathcal{X}$ does not get any approval from the votes that Bob generates. Now, the k-approval score of the candidate (x_i, i) is at least $(k+1)\frac{\varepsilon n}{2}$, whereas every other candidate gets at most $\frac{k\varepsilon n}{2}$ many approvals. Hence, (x_i, i) is the unique $\frac{\varepsilon}{5}$-winner. If $k > \frac{1}{\varepsilon}$ then, Bob generates $\frac{n}{2}$ votes in such a way that every candidate in $\mathcal{X}$ gets $\frac{n}{2}$ many approvals and every candidate in $[\frac{1}{\varepsilon}] \times [\frac{k}{\varepsilon}] \setminus \mathcal{X}$ gets at most $(k - \frac{1}{\varepsilon})\frac{n}{2}\frac{1}{k/\varepsilon^2 - 1/\varepsilon} \leq \frac{n}{2}\varepsilon^2$ many approvals from the votes that Bob generates. Here again the k-approval score of the candidate (x_i, i) is at least $(1 + \varepsilon)\frac{n}{2}$, where as the k-approval score of every other candidate is at most $\frac{\varepsilon n}{2}$. Hence, (x_i, i) is the unique $\frac{\varepsilon}{5}$-winner. $\qquad\square$

For the generalized plurality voting rule, we provide a $\Omega(\frac{1}{\sqrt{\varepsilon}}\log m)$ space complexity lower bound, again by reducing from AUGMENTED-INDEXING$_{m,\frac{1}{\sqrt{\varepsilon}}}$. This bound is better than the lower bound of Theorem 15 when m is exponentially larger compared to $\frac{1}{\varepsilon}$.

Theorem 18. *Suppose the number of candidates m is at least $\frac{1}{\sqrt{\varepsilon}}$. Any one pass (ε,δ)–WINNER DETERMINATION algorithm for the generalized plurality rule must use $\Omega(\frac{1}{\sqrt{\varepsilon}}\log m)$ bits of memory, for every $1-\delta > \frac{3\varepsilon}{2}$.*

Proof. We prove the result for $(\frac{\varepsilon}{5},\delta)$–WINNER DETERMINATION problem. Consider the AUGMENTED-INDEXING$_{m,\frac{1}{\sqrt{\varepsilon}}}$ problem where Alice is given a string $x = (x_1,\cdots,x_{\frac{1}{\sqrt{\varepsilon}}}) \in [m]^{\frac{1}{\sqrt{\varepsilon}}}$ and Bob is given an integer $i \in [\frac{1}{\sqrt{\varepsilon}}]$ and $(x_1,\cdots,x_{i-1})$. The candidate set of our election is $[m]\times[\frac{1}{\sqrt{\varepsilon}}]$. The votes would be such that the only $\frac{\varepsilon}{5}$–winner will be the candidate (x_i,i), thereby proving the result. Thus Bob can output x_i correctly whenever our $(\frac{\varepsilon}{5},\delta)$–WINNER DETERMINATION algorithm outputs correctly. Alice generates $(\frac{1}{\sqrt{\varepsilon}}-j)\varepsilon n$ many approvals for candidate (x_j,j), for every $j < \frac{1}{\sqrt{\varepsilon}}$. Alice now sends the memory content of the algorithm. Bob resumes the run of the algorithm by generating $(\frac{1}{\sqrt{\varepsilon}}-j)\varepsilon n$ many approvals for candidate (x_j,j), for every $j < i$. Notice that, the only $\frac{\varepsilon}{5}$-winner is the candidate (x_i,i). Now the space complexity lower bound follows from Lemma 2. $\qquad\square$

The space complexity lower bound in Theorem 15 for the plurality voting rule matches with the upper bound of Theorem 3, when $\frac{1}{\varepsilon} \le m \le \frac{1}{\varepsilon^{O(1)}}$. For the case when $m \le \frac{1}{\varepsilon}$, we now show a matching space complexity lower bound for the plurality voting rule. We prove this result by exhibiting a reduction from the MAX-SUM$_{\frac{1}{\varepsilon},m}$ problem.

Theorem 19. *Assume that the number of candidates m is at most $\frac{1}{\varepsilon}$. Then any one pass (ε,δ)–WINNER DETERMINATION algorithm for the plurality, generalized plurality, approval, k-approval for $k = O(m^\lambda)$, for any $\lambda \in [0,1)$, maximin, Copeland, Bucklin, plurality with run off voting rules must use $\Omega(m\log\frac{1}{\varepsilon})$ bits of memory.*

Proof. First, let us prove the result for the plurality voting rule. Suppose we have a one pass (ε,δ)–WINNER DETERMINATION algorithm for the plurality election which uses $s(n,\varepsilon)$ bits of memory. Consider the communication problem MAX-SUM$_{\frac{1}{\varepsilon},m}$. Let the inputs to Alice and Bob in the MAX-SUM$_{\frac{1}{\varepsilon},m}$ instance be $x = (x_1,x_2,\cdots,x_m) \in [\frac{1}{\varepsilon}]^m$ and $y = (y_1,y_2,\cdots,y_m) \in [\frac{1}{\varepsilon}]^m$ respectively. The candidate set of the election is $[m]$. Alice generates x_i many plurality vote for the candidate i, for every $i \in [m]$. Alice now sends the memory content of the algorithm to Bob. Bob resumes the run of the algorithm by generating y_i many plurality votes for the candidate i, for every $i \in [m]$. Suppose $i = \arg\max_{j\in[m]}\{x_j+y_j\}$ (recall from Definition 6 that there exist unique element i that maximizes x_i+y_i) and $\ell \ne \arg\max_{j\in[m]}\{x_j+y_j\}$. Then we have the following:

$$\frac{(x_i+y_i)-(x_\ell+y_\ell)}{\sum_{j\in[m]}(x_j+y_j)} \ge \frac{\varepsilon}{2m} \ge \frac{\varepsilon^2}{2}$$

The first inequality follows from the fact that $(x_i+y_i)-(x_\ell+y_\ell) \ge 1$ and $\sum_{j\in[m]}x_j+y_j \le \frac{2m}{\varepsilon}$. The second inequality follows from the assumption that $m \le \frac{1}{\varepsilon}$. Hence, whenever the $(\frac{\varepsilon^2}{5},\delta)$–WINNER DETERMINATION algorithm outputs an $\frac{\varepsilon^2}{5}$-winner, Bob also outputs correctly in the MAX-SUM$_{\frac{1}{\varepsilon},m}$ problem instance.

For the other voting rules, the idea is the same as above: we will generate votes in such a way that ensures that the candidate i wins if $i = \arg\max_{j \in [m]} \{x_j + y_j\}$ by a margin of at least one. Below, we only specify the votes to be generated for other voting rules.

- **Generalized plurality, approval:** Follows immediately from the fact that every plurality election is a valid generalized plurality and approval election too.

- k-**approval for** $k = O(m^\lambda)$, **for any** $\lambda \in [0, 1)$: Alice (respectively Bob) generates x_i (respectively y_i) many votes such that candidate i gets x_i many approvals and the rest $(k-1)x_i$ many approvals are equally distributed among other $m - 1$ candidates.

- **Borda, maximin, Copeland, Bucklin, plurality with run off:** Alice (respectively Bob) generates x_i (respectively y_i) many votes of the form $i \succ \overrightarrow{\mathcal{C}_{-i}}$ and another x_i (respectively y_i) many votes of the form $i \succ \overleftarrow{\mathcal{C}_{-i}}$, where $\overrightarrow{\mathcal{C}_{-i}}$ is an arbitrary but fixed order of the candidates in $\mathcal{C} \smallsetminus \{i\}$ and $\overleftarrow{\mathcal{C}_{-i}}$ is the reverse order of $\overrightarrow{\mathcal{C}_{-i}}$. $\hspace{2em} \square$

Now we show space complexity lower bounds that depend on the number of votes n. The result below is obtained by reducing from the GREATER-THAN$_n$ problem. The lower bound is tight in the number of votes n.

Theorem 20. *Any one pass (ε, δ)–WINNER DETERMINATION algorithm for the plurality voting rule must use $\Omega(\log \log n)$ memory bits, even if the number of candidates is only 2, for every $\delta < \frac{1}{4}$. The same applies for generalized plurality, scoring rules, maximin, Copeland, Bucklin, and plurality with run off voting rules.*

Proof. Suppose we have a one pass (ε, δ)–WINNER DETERMINATION algorithm for the plurality election which uses $s(n)$ bits of space. Using this algorithm, we will show a communication protocol for the GREATER-THAN$_n$ problem whose communication coplexity is $s(2^n)$ thereby proving the statement. The candidate set is $\{0, 1\}$. Alice generates a stream of 2^x many plurality votes for the candidate 1. Alice now sends the memory content of the algorithm. Bob resumes the run of the algorithm by generating a stream of 2^y many plurality votes for the candidate 0. If $x > y$ then the candidate 1 is the only ε-winner; whereas if $x < y$ then the candidate 0 is the only ε-winner.

For elections with two candidates, generalized plurality, scoring rules, maximin, Copeland, Bucklin, and plurality with run off voting rules are same as the plurality voting rule. $\hspace{1em} \square$

5   Conclusions and Future Work

In this work, we studied the space complexity for determining approximate winners in the setting where votes are inserted continually into a data stream. We showed that allowing randomization and approximation indeed allows for much more space-efficient algorithms. Moreover, our bounds are tight in certain parameter ranges.

The most immediate open question is to close the gaps between the upper and lower bounds. In particular, even for plurality, the dependence on m and ε is not tight when m is large. Also, for the other voting rules, are there more sophisticated algorithms which improve our upper bounds? In a different vein, it may be interesting to implement these streaming algorithms for use in practice (say, for participatory democracy experiments or for online social networks) and investigate how they perform. Finally, instead of having the algorithm be passive, could we improve performance by having the algorithm actively query the voters as they appear in the stream?

References

[syn] Synapp. `http://synapp.stanford.edu/`. Accessed: 2015-07-21.

[wid] Widescope. `http://widescope.stanford.edu/`. Accessed: 2015-07-21.

[3] Alon, N., Matias, Y., and Szegedy, M. (1999). The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137 – 147.

[4] Bar-Yossef, Z., Jayram, T., Kumar, R., and Sivakumar, D. (2002). An information statistics approach to data stream and communication complexity. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pages 209–218. IEEE.

[5] Boyer, R. S. and Moore, J. S. (1991). MJRTY- a fast majority vote algorithm. In Boyer, R. S., editor, *Automated Reasoning*, volume 1 of *Automated Reasoning Series*, pages 105–117. Springer Netherlands.

[6] Braverman, V., Ostrovsky, R., and Zaniolo, C. (2009). Optimal sampling from sliding windows. In *Proceedings of the Twenty-eighth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '09, pages 147–156, New York, NY, USA. ACM.

[7] Caragiannis, I. and Procaccia, A. D. (2011). Voting almost maximizes social welfare despite limited communication. *Artificial Intelligence*, 175(910):1655 – 1671.

[8] Chakrabarti, A., Khot, S., and Sun, X. (2003). Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *Computational Complexity, 2003. Proceedings. 18th IEEE Annual Conference on*, pages 107–117. IEEE.

[9] Charikar, M., Chen, K., and Farach-Colton, M. (2004). Finding frequent items in data streams. *Theoretical Computer Science*, 312(1):3–15.

[10] Conitzer, V. and Sandholm, T. (2005). Communication complexity of common voting rules. In *Proceedings of the 6th ACM conference on Electronic commerce*, pages 78–87. ACM.

[11] Cormode, G. and Hadjieleftheriou, M. (2008). Finding frequent items in data streams. *Proceedings of the VLDB Endowment*, 1(2):1530–1541.

[12] Cormode, G. and Muthukrishnan, S. (2005). An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75.

[13] Datar, M., Gionis, A., Indyk, P., and Motwani, R. (2002). Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813.

[14] Demaine, E. D., López-Ortiz, A., and Munro, J. I. (2002). Frequency estimation of internet packet streams with limited space. In *ESA*, pages 348–360. Springer.

[15] Dey, P. and Bhattacharyya, A. (2015). Sample complexity for winner prediction in elections. In *Proceeding of the 14th International Conference on Autonomous Systems and Multiagent Systems (AAMAS-15)*.

[16] Ergün, F., Jowhari, H., and Sağlam, M. (2010). Periodicity in streams. In *APPROX and RANDOM*, pages 545–559. Springer.

[17] Fischer, M. J. and Salzburg, S. L. (1982). Finding a majority among n votes: Solution to problem 81-5. *Journal of Algorithms*, 3(4):376–379.

[18] Flajolet, P. (1985). Approximate counting: a detailed analysis. *BIT Numerical Mathematics*, 25(1):113–134.

[19] Gronemeier, A. and Sauerhoff, M. (2009). Applying approximate counting for computing the frequency moments of long data streams. *Theory Comput. Syst.*, 44(3):332–348.

[20] Henzinger, M. R., Raghavan, P., and Rajagopalan, S. (1999). External memory algorithms. chapter Computing on Data Streams, pages 107–118. American Mathematical Society, Boston, MA, USA.

[21] Jowhari, H., Sağlam, M., and Tardos, G. (2011). Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '11, pages 49–58, New York, NY, USA. ACM.

[22] Karp, R. M., Shenker, S., and Papadimitriou, C. H. (2003). A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems (TODS)*, 28(1):51–55.

[23] Kremer, I., Nisan, N., and Ron, D. (1999). On randomized one-round communication complexity. *Computational Complexity*, 8(1):21–49.

[24] Kushilevitz, E. and Nisan, N. (1997). *Communication Complexity*. Cambridge University Press, New York, NY, USA.

[25] Lee, D. T., Goel, A., Aitamurto, T., and Landemore, H. (2014). Crowdsourcing for participatory democracies: Efficient elicitation of social choice functions. In *Proceedings of the Seconf AAAI Conference on Human Computation and Crowdsourcing, HCOMP 2014, November 2-4, 2014, Pittsburgh, Pennsylvania, USA*.

[26] Lesser, V., Ortiz Jr, C. L., and Tambe, M. (2012). *Distributed sensor networks: A multiagent perspective*, volume 9. Springer Science & Business Media.

[27] Manku, G. S. and Motwani, R. (2002). Approximate frequency counts over data streams. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 346–357. VLDB Endowment.

[28] Metwally, A., Agrawal, D., and El Abbadi, A. (2005). Efficient computation of frequent and top-k elements in data streams. In *Database Theory-ICDT 2005*, pages 398–412. Springer.

[29] Miltersen, P. B., Nisan, N., Safra, S., and Wigderson, A. (1998). On data structures and asymmetric communication complexity. *J. Comput. Syst. Sci.*, 57(1):37–49.

[30] Misra, J. and Gries, D. (1982). Finding repeated elements. *Sci. Comput. Program.*, 2(2):143–152.

[31] Morris, R. (1978). Counting large numbers of events in small registers. *Commun. ACM*, 21(10):840–842.

[32] Muthukrishnan, S. (2005). *Data streams: Algorithms and applications*. Now Publishers Inc.

[33] Nelson, J. (2012). Sketching and streaming algorithms for processing massive data. *XRDS: Crossroads, The ACM Magazine for Students*, 19(1):14–19.

[34] Smirnov, D. (1988). Shannon's information methods for lower bounds for probabilistic communication complexity. Master's thesis, Moscow University.

[35] Vitter, J. S. (1985). Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57.

[36] Xia, L. (2012). Computing the margin of victory for various voting rules. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, pages 982–999. ACM.

[37] Yao, A. C.-C. (1979). Some complexity questions related to distributive computing (preliminary report). In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 209–213. ACM.