Element Distinctness, Frequency Moments, and Sliding Windows

Paul Beame*
Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350
beame@cs.washington.edu

Raphaël Clifford†
Department of Computer Science
University of Bristol
Bristol BS8 1UB, United Kingdom
clifford@cs.bris.ac.uk

Widad Machmouchi*
Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350
widad@cs.washington.edu

Abstract— We derive new time-space tradeoff lower bounds and algorithms for exactly computing statistics of input data, including frequency moments, element distinctness, and order statistics, that are simple to calculate for sorted data. In particular, we develop a randomized algorithm for the element distinctness problem whose time T and space S satisfy $T \in \tilde{O}(n^{3/2}/S^{1/2})$, smaller than previous lower bounds for comparison-based algorithms, showing that element distinctness is strictly easier than sorting for randomized branching programs. This algorithm is based on a new time- and space-efficient algorithm for finding all collisions of a function f from a finite set to itself that are reachable by iterating f from a given set of starting points.

We further show that our element distinctness algorithm can be extended at only a polylogarithmic factor cost to solve the element distinctness problem over sliding windows [18], where the task is to take an input of length $2n-1$ and produce an output for each window of length n, giving n outputs in total.

In contrast, we show a time-space tradeoff lower bound of $T \in \Omega(n^2/S)$ for randomized multi-way branching programs, and hence standard RAM and word-RAM models, to compute the number of distinct elements, F_0, over sliding windows. The same lower bound holds for computing the low-order bit of F_0 and computing any frequency moment F_k for $k \neq 1$. This shows that frequency moments $F_k \neq 1$ and even the decision problem $F_0 \bmod 2$ are strictly harder than element distinctness. We provide even stronger separations on average for inputs from $[n]$.

We complement this lower bound with a $T \in \tilde{O}(n^2/S)$ comparison-based deterministic RAM algorithm for exactly computing F_k over sliding windows, nearly matching both our general lower bound for the sliding-window version and the comparison-based lower bounds for a single instance of the problem. We also consider the computations of order statistics over sliding windows.

Keywords-Time-space tradeoffs, Frequency moments, Branching programs, Collision finding

1. Introduction

Problems related to computing elementary statistics of input data have wide applicability and utility. Despite their usefulness, there are surprising gaps in our knowledge about the best ways to solve these simple problems, particularly in the context of limited storage space. Many of these elementary statistics can be easily calculated if the input data is already sorted but sorting the data in space S, requires time $T \in \Omega(n^2/S)$ [12], [9], a bound matched by the best comparison algorithms [26] for all $S \in O(n/\log n)$. It has not been clear whether exactly computing elementary statistical properties, such as frequency moments (e.g. F_0, the number of distinct elements in the input) or element distinctness (ED, whether or not F_0 equals the input size) are as difficult as sorting when storage is limited.

The main approach to proving time-space tradeoff lower bounds for problems in P has been to analyze their complexity on (multi-way) *branching programs*. (The input is assumed to be stored in read-only memory and the output in write-only memory and neither is counted towards the space used by any algorithm. The multi-way branching program model simulates both Turing machines and standard RAM models that are unit-cost with respect to time and log-cost with respect to space.) An important method for this analysis was introduced by Borodin and Cook for sorting [12] and has since been extended and generalized to randomized computation of a number of other important multi-output problems (e.g., [33], [2], [3], [9], [24], [29]). Unfortunately, the techniques of [12] yield only trivial bounds for problems with single outputs such as F_0 or ED.

Element distinctness has been a particular focus of lower bound analysis. The first time-space tradeoff lower bounds for the problem apply to structured algorithms. Borodin et al. [13] gave a time-space tradeoff lower bound for computing ED on *comparison* branching programs of $T \in \Omega(n^{3/2}/S^{1/2})$ and, since $S \geq \log_2 n$, $T \in \Omega(n^{3/2}\sqrt{\log n}/S)$. Yao [32] improved this to a near-optimal $T \in \Omega(n^{2-\epsilon(n)}/S)$, where $\epsilon(n) = 5/(\ln n)^{1/2}$. Since these lower bounds apply to the average case for randomly ordered inputs, by Yao's lemma, they also apply to randomized comparison branching programs. These bounds also trivially apply to all frequency moments since, for $k \neq 1$, $ED(x) = n$ iff $F_k(x) = n$. This near-quadratic lower bound seemed to suggest that the complexity of ED and F_k should closely track that of sorting.

For multi-way branching programs, Ajtai [4] showed that any linear time algorithm for ED must consume linear space. Moreover, when S is $n^{o(1)}$, Beame et al. [11] showed a $T \in \Omega(n\sqrt{\log(n/S)/\log\log(n/S)})$ lower bound for computing ED. This is a long way from the comparison branching program lower bound and there has not

*Research supported by NSF grants CCF-1217099 and CCF-0916400
†Research supported by the EPSRC. This work was done while the author was visiting the University of Washington.

0272-5428/13 $26.00 © 2013 IEEE
DOI 10.1109/FOCS.2013.39

been much prospect for closing the gap since the largest lower bound known for multi-way branching programs computing *any* single-output problem in P is only $T \in \Omega(n \log((n \log n)/S))$.

We show that this gap between sorting and element distinctness cannot be closed. More precisely, we give a randomized multi-way branching program algorithm that for any space bound $S \in [c \log n, n]$ computes ED in time $T \in \tilde{O}(n^{3/2}/S^{1/2})$ where $\tilde{O}$ suppresses polylogarithmic factors in n, significantly beating the lower bound that applies to comparison-based algorithms. Our algorithm for ED is based on an extension of Floyd's cycle-finding algorithm [21] (more precisely, its variant, Pollard's rho algorithm [27]). Pollard's rho algorithm finds the unique collision reachable by iterating a function $f : [n] \to [n]$ from a single starting location in time proportional to the size of the reachable set, using only a constant number of pointers. Variants of this algorithm have been used in cryptographic applications to find collisions in functions that supposedly behave like random functions [15], [30], [25].

More precisely, our new ED algorithm is based on a new deterministic extension of Floyd's algorithm to find *all* collisions of a function $f : [n] \to [n]$ reachable by iterating f from any one of a set of k starting locations, using only $O(k)$ pointers and using time roughly proportional to the size of the reachable set. Motivated by cryptographic applications, [31] previously considered this problem for the special case of random functions and suggested a method using 'distinguished points', though the only analysis they gave was heuristic and incomplete. Our algorithm, developed independently, uses a different method, applies to arbitrary functions, and has a fully rigorous analysis.

Does the gap between ED and sorting extend to frequency moment computation? Given the general difficulty of obtaining strong lower bounds for single-output functions, we consider the relative complexity of computing many copies of each of the functions at once and apply techniques for multi-output functions to make the comparison. Since we want to retain a similar input size to that of our original problems, we need to evaluate them on overlapping inputs.

Evaluating the same function on overlapping inputs occurs as a natural problem in time series analyis when it is useful to know the value of a function on many different intervals or *windows* within a sequence of values or updates, each representing the recent history of the data at a given instant. Such computations have been termed *sliding-window* computations for the associated functions [18]. In particular, we consider inputs of length $2n - 1$ where the sliding-window task is to compute the function for each window of length n, giving n outputs in total. We write $F^{\boxplus n}$ to denote this sliding-window version of a function F.

Many natural functions have been studied for sliding windows including entropy, finding frequent symbols, frequency moments and order statistics, which can be computed approximately in small space using randomization even in one-pass data stream algorithms [18], [8], [7], [22], [23], [16], [14]. Approximation is required since exactly computing these values in this online model can easily be shown to require large space.

We show that computing ED over n sliding windows only incurs a polylogarithmic overhead in time and space versus computing a single copy of ED. In particular, we can extend our randomized multi-way branching program algorithm for ED to yield an algorithm for $ED^{\boxplus n}$ that for space $S \in [c \log n, n]$ runs in time $T \in \tilde{O}(n^{3/2}/S^{1/2})$.

In contrast, we prove strong time-space lower bounds for computing the sliding-window version of any frequency moment F_k for $k \neq 1$: The time T and space S to compute $F_k^{\boxplus n}$ must satisfy $T \in \Omega(n^2/S)$ and $S \geq \log n$. The bounds are proved directly for randomized multi-way branching programs which imply lower bounds for the standard RAM and word-RAM models, as well as for the data stream models discussed above. We show that the same lower bound holds for computing the parity of the number of distinct elements, $F_0 \bmod 2$, in each window. This formally proves that sliding-window $F_0 \bmod 2$ is strictly harder than sliding-window ED and suggests that for proving strong complexity lower bounds, $F_0 \bmod 2$ may be a better choice to analyze than ED.

Our lower bounds for frequency moment computation hold for randomized algorithms even with small success probability $2^{-O(S)}$ and for the average time and space used by deterministic algorithms on inputs in which the values are independently and uniformly chosen from $[n]$. (For comparison with the latter average case results, it is not hard to show that over the same input distribution ED can be solved with $\overline{T} \in \tilde{O}(n/\overline{S})$ and our reduction shows that this can be extended to $\overline{T} \in \tilde{O}(n/\overline{S})$ bound for $ED^{\boxplus n}$ on this input distribution.)

We complement our lower bound with a comparison-based RAM algorithm for any $F_k^{\boxplus n}$ that has $T \in \tilde{O}(n^2/S)$, showing that this is nearly an asymptotically tight bound for RAM algorithms. Since our algorithm for computing $F_k^{\boxplus n}$ is comparison-based, the comparison lower bound for F_k implied by [32] is not far from matching our algorithm even for a single instance of F_k. In the full paper we also show that quantum algorithms for $F_k^{\boxplus n}$ require only $\tilde{O}(n^{3/2})$ time when the space is $O(\log n)$.

The range of relationships between the complexity of computing a function F and that of computing $F^{\boxplus n}$ is illustrated by considering problems of computing the t^{th} order statistic in each window. In the case of $t = n$ (maximum) or $t = 1$ (minimum) we show that computing these properties over sliding windows can be done by a comparison based algorithm in $O(n \log n)$ time and only $O(\log n)$ bits of space so there is very little growth in complexity. In contrast, we show that a $T \in \Omega(n^2/S)$ lower bound holds when $t = \alpha n$ for any fixed $0 < \alpha < 1$. Even for algorithms that only use comparisons, the expected time for errorless randomized algorithms to find the median in a single window is $\overline{T} \in \Theta(n \log\log_S n)$ [17]; hence, these problems have a dramatic increase in complexity over sliding windows.

Related work: Sliding-windows versions of problems have been considered in the context of online and approximate computation but the only instance of which we are aware that has considered sliding windows for exact offline computation is a lower bound for generalized string matching due to Abrahamson [2]. This shows that for any fixed string $y \in [n]^n$ with n distinct values, $H_y^{\boxplus n}$ requires $T \cdot S \in \Omega(n^2/\log n)$ where decision problem $H_y(x)$ is 1 if and only if the Hamming distance between x and y is n. This bound is an $\Omega(\log n)$ factor smaller than our lower bound for sliding-window $F_0 \bmod 2$.

Frequency Moments, Element Distinctness, and Order Statistics: Let $a = a_1a_2\ldots a_n \in D^n$ for some finite set D. We define the k^{th} *frequency moment* of a, $F_k(a)$, as $F_k(a) = \sum_{i\in D} f_i^k$, where f_i is the frequency (number of occurrences) of symbol i in the string a and D is the set of symbols that occur in a. Therefore, $F_0(a)$ is the number of distinct symbols in a and $F_1(a) = |a|$ for every string a. The *element distinctness* problem is a decision problem defined as: $ED(a) = 1$ if $F_0(a) = |a|$ and 0 otherwise. We write ED_n for the ED function restricted to inputs a with $|a| = n$. The t^{th} *order statistic* of a, O_t, is the t^{th} smallest symbol in a. Therefore O_n is the maximum of the symbols of a and $O_{\lceil \frac{n}{2} \rceil}$ is the median.

Branching programs: Let D and R be finite sets and n and m be positive integers. A *D-way branching program* is a connected rooted directed acyclic graph and possibly many *sink nodes*. Each non-sink node is labeled with an input index$\in [n]$ and every edge is labeled with a symbol from D, which corresponds to the value of the input indexed at the originating node. In order not to count the space required for outputs, as is standard for the multi-output problems [12], we assume that each edge can be labeled by some set of output assignments. For a directed path π in a branching program, we call the set of indices of symbols queried by π the *queries* of π, denoted by Q_π; we denote the *answers* to those queries by $A_\pi : Q_\pi \to D$ and the outputs produced along π as a *partial* function $Z_\pi : [m] \to R$. A branching program computes a function $f : D^n \to R^m$ by following the path given by the query outcomes in the obvious way.

A branching program B is computes a function f if for every $x \in D^n$, the output of B on x, denoted $B(x)$, is equal to $f(x)$. The *computation* of B on x is a directed path, denoted $\pi_B(x)$, from the source to a sink in B whose queries to the input are consistent with x. The time T of a branching program is the length of the longest path from the source to a sink and the space S is the logarithm base 2 of the number of the nodes in the branching program. Therefore, $S \geq \log T$ where we write $\log x$ to denote $\log_2 x$.

A *randomized* branching program $\mathcal{B}$ is a probability distribution over deterministic branching programs with the same input set. $\mathcal{B}$ computes a function f with error at most η if for every input $x \in D^n$, $\Pr_{B\sim\mathcal{B}}[B(x) = f(x)] \geq 1-\eta$. The time (resp. space) of a randomized branching program is the maximum time (resp. space) of a deterministic branching program in the support of the distribution.

While our lower bounds apply to randomized branching programs, which allow the strongest non-explicit randomness, our randomized algorithms for element distinctness will only require a weaker notion, *input randomness*, in which the random string r is given as an explicit input to a RAM algorithm. For space-bounded computation, it would be preferable to only require the random bits to be available online as the algorithm proceeds.

As is usual in analyzing randomized computation via Yao's lemma, we also will consider complexity under distributions μ on the input space D^n. A branching program B *computes f under μ with error at most η* iff $B(x) = f(x)$ for all but an η-measure of $x \in D^n$ under distribution μ.

2. Element Distinctness and Small-Space Collision-Finding

2.1. Small-space collision-finding with many sources

Our approach for solving the element distinctness problem has at its heart a novel extension of Floyd's small space "tortoise and hare" cycle-finding algorithm [21]. Given a start vertex v in a finite graph $G = (V, E)$ of outdegree 1, Floyd's algorithm finds the unique cycle in G that is reachable from v. The out-degree 1 edge relation E can be viewed as a set of pairs $(u, f(u))$ for a function $f : V \to V$. Floyd's algorithm, more precisely, stores only two values from V and finds the smallest s and $\ell > 0$ and vertex w such such that $f^s(v) = f^{s+\ell}(v) = w$ using only $O(s + \ell)$ evaluations of f.

We say that vertices $u \neq u' \in V$ are *colliding* iff $f(u) = f(u')$ and call $v = f(u) = f(u')$ a collision. Floyd's algorithm for cycle-finding can also be useful for finding collisions since in many instances the starting vertex v is not on a cycle and thus $s > 0$. In this case $i = f^{s-1}(v) \neq j = f^{s+\ell-1}(v)$ satisfy $f(i) = f(j) = w$, which is a collision in f, and the iterates of f produce a ρ shape (see Figure 1a). These colliding points may be found with minimal cost by also storing the previous values of each of the two pointers as Floyd's algorithm proceeds. The ρ shape inspired the name of Pollard's rho algorithm for factoring [27] and solving discrete logarithm problems [28] and the application of Floyd's cycle finding algorithm to collision-finding usually goes by this name.

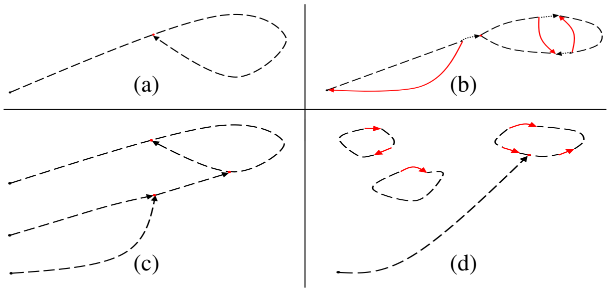

Figure 1: Collision-finding with multiple sources

There is considerable work, particularly for cryptographic applications, on collision-finding algorithms that use larger space than Floyd's algorithm and improve the constant factors in the number of edges that must be traversed (function evaluations) to find a collision reachable from a single starting point (c.f. [15], [30], [25] and references therein).

Motivated by our application of efficiently solving element distinctness, we examine the time and space complexity of finding *all* colliding vertices, along with their predecessors, in the subgraph reachable from a possibly large set of k starting vertices, not just from a single start vertex. We will show how to do this using storage equivalent to only $O(k)$ elements of V and time roughly proportional to the size of the subgraph reachable from this set of starting vertices. Note that the obvious approach of running k independent copies of Floyd's algorithm in parallel from each of the start vertices does not solve this problem since it may miss collisions between different parallel branches (see Figure 1c), and it may also traverse large regions of the subgraph many times.

In [31], van Oorschot and Wiener gave a deterministic parallel algorithm for finding all collisions of a *random* function using k processors, which keeps a record of visits to predetermined vertices ('distinguished points') allowing the separate processes to determine quickly if they are a previously explored path. They gave a heuristic argument suggesting a bound of $O(n^{3/2}/k^{1/2})$ function evaluations, though they were unable to supply a rigorous argument. Our method is very different in detail and developed independently; we provide a fully rigorous analysis that roughly matches the heuristic bound of [31] for the related problem of collision-finding on random hash functions applied to worst-case inputs for element distinctness:

For $v \in V$, define $f^*(v) = \{f^i(v) \mid i \geq 0\}$ to be the set of vertices reachable from v and $f^*(U) = \bigcup_{v \in U} f^*(v)$ for $U \subseteq V$.

Theorem 2.1. *There is an $O(k \log n)$ space deterministic algorithm* $\textsc{Collide}_k$ *that, given $f : V \to V$ for a finite set V and $K = \{v_1, \ldots, v_k\} \subseteq V$, finds all pairs $(v, \{u \in f^*(K) | f(u) = v\})$ and runs in time $O(|f^*(K)|\sqrt{\log k / \log\log k}\min\{k, \log n\})$.*

Proof: We first describe the algorithm $\textsc{Collide}_k$: In addition to the original graph and the collisions that it finds, this algorithm maintains a *redirection list* $R \subset V$ of size $O(k)$ vertices that it provisionally redirects to map to a different location. For each vertex in R it stores the name of the new vertex to which it is directed. We maintain a separate list L of all vertices from which an edge of G has been redirected away and the original vertices that point to them.

```
COLLIDE_k:
Set $R = \emptyset$.
For $j = 1, \ldots, k$ do:
  1) Execute Floyd's algorithm starting with vertex $v_j$ on
     the graph $G$ using the redirected out-edges for nodes
     from the redirection list $R$ instead of $f$.
  2) If the cycle found does not include $v_j$, there must be a
     collision.
     a) If this collision is in the graph $G$, report the collision
        $v$ as well as the colliding vertices $u$ and $u'$, where
        $u'$ is the predecessor of $v$ on the cycle and $u$ is the
        predecessor of $v$ on the path from $v_j$ to $v$.
     b) Add $u$ to the redirection list $R$ and redirect it to
        vertex $v_j$.
  3) Traverse the cycle again to find its length and choose
     two vertices $w$ and $w'$ on the cycle that are within 1 of
     half this length apart. Add $w$ and $w'$ to the redirection
     list, redirecting $w$ to $f(w')$ and $w'$ to $f(w)$. This will
     split the cycle into two parts, each of roughly half its
     original length.
```

The redirections for a single iteration of the algorithm are shown in Figure 1b. The general situation for later iterations in the algorithm is shown in Figure 1d.

In each iteration of the loop there is at most one vertex v where collisions can occur and at most 3 vertices are added to the redirection list. Moreover, after each iteration, the set of vertices reachable from vertices $v_1, \ldots, v_j$ appear in a collection of disjoint cycles of the redirected graph. Each iteration of the loop traverses at most one cycle and every cycle is roughly halved each time it is traversed.

In order to store the redirection list R, we use a dynamic dictionary data structure of $O(k \log n)$ bits that supports insert and search in $O(\log k)$ time per access or insertion. We can achieve this using balanced binary search trees and we can improve the bound to $O(\sqrt{\log k / \log\log k})$ using exponential trees [6]. Before following an edge (i.e., evaluating f), the algorithm will first check list R to see if it has been redirected. Hence each edge traversed costs $O(\log k)$ time (or $O(\sqrt{\log k / \log\log k})$ using exponential trees). Since time is measured relative to the size of the reachable set of vertices, the only other extra cost is that of re-traversing previously discovered edges. Since all vertices are maintained in cycles and each traversal of a cycle roughly halves its length, each edge found can be traversed at most $O(\min\{k, \log n\})$ times. ■

2.2. A randomized $T^2S\in \tilde{O}(n^3)$ element distinctness algorithm

We will use collision-finding for our element distinctness algorithm. The vertex set V will be the set of indices $[n]$, and the function f will be given by $f_{x,h}(i) = h(x_i)$ where h is a (random) hash function that maps $[m]$ to $[n]$.

Observe that if we find $i \neq j$ such that $f_{x,h}(i) = f_{x,h}(j)$ then either

- $x_i = x_j$ and hence $\mathrm{ED}(x) = 0$, or
- we have found a collision in h: $x_i \neq x_j$ but $h(x_i) = h(x_j)$; we call x_i and x_j "pseudo-duplicates" in this latter case.

Given a parameter k, on input x our randomized algorithm will repeatedly choose a random hash function h and a random set K of roughly k starting points and then call the $\textsc{Collide}_k$ algorithm given in Theorem 2.1 on K using the function $f = f_{x,h}$ and check the collisions found to determine whether or not there is a duplicate among the elements of x indexed by $f^*_{x,h}(K)$. The space bound S of this algorithm will be $O(k \log n)$.

The running time of $\textsc{Collide}_k$ depends on $|f^*_{x,h}(K)|$, which in turn is governed by the random choices of h and K and may be large.

Since $f^*_{x,h}(K)$ is also random, we also need to argue that if $ED(x) = 0$, then there is a reasonable probability that a duplicate in x will be found among the indices in $f^*_{x,h}(K)$. The following two lemmas analyze these issues.

Lemma 2.2. *Let $x \in [m]^n$. For $h : [m] \to [n]$ chosen uniformly at random and for $K \subseteq [n]$ selected by uniformly and independently choosing $2 \leq k \leq n/32$ elements of $[n]$ with replacement, $\Pr[|f^*_{x,h}(K)| \leq 2\sqrt{kn}] \geq 8/9$.*

Lemma 2.3. *Let $x \in [m]^n$ be such that $ED(x) = 0$. Then for $h : [m] \to [n]$ chosen uniformly at random and for $K \subseteq [n]$ selected by uniformly and independently choosing $2 \leq k \leq n/32$ elements of $[n]$ with replacement, then $\Pr[|f^*_{x,h}(K)| \leq 2\sqrt{kn}$ and $\exists i \neq j \in f^*_{x,h}(K)$ s.t. $x_i = x_j]$ is at least $k/(18n)$.*

These lemmas, together with the properties of $\textsc{Collide}_k$ yield the following theorem.

Theorem 2.4. *For any $\epsilon > 0$, and any S with $c \log n \leq S \leq n/32$ for some constant $c > 0$, there is a randomized RAM algorithm with input randomness computing ED_n with 1-sided error (false positives) at most ϵ, that uses space S and time $T \in O(\frac{n^{3/2}}{S^{1/2}} \log^{5/2} n \log(1/\epsilon))$. Further, when $S \in O(\log n)$, we have $T \in O(n^{3/2} \log(1/\epsilon))$.*

Proof: Choose $k \geq 2$ such that the space usage of $\textsc{Collide}_k$ on $[n]$ is at most $S/2$. Therefore $k \in \Omega(S/\log n)$. The algorithm is as follows:
On input x, run $(18n/k)\log(1/\epsilon)$ independent runs of $\textsc{Collide}_k$ on different $f_{x,h}$, each with independent random choices of hash functions h and independent choices, K, of k starting indices, and each with a run-time cut-off bounding the number of explored vertices of $f^*_{x,h}(K)$ at $t^* = 2\sqrt{kn}$. On each run, check if any of the collisions found is a duplicate in x, in which case output $ED(x) = 0$ and halt. If none are found in any round then output $ED(x) = 1$.

The algorithm will never incorrectly report a duplicate in a distinct x and by Lemma 2.3, each run has a probability of at least $k/(18n)$ of finding a duplicate in an input x such that $ED(x) = 0$ so the probability of failing to find a duplicate in $(18n/k)\log(1/\epsilon)$ rounds is at most ϵ.

Using Theorem 2.1 each run of our bounded version of $\textsc{Collide}_k$, requires runtime $O(\sqrt{kn} \log k \min\{k, \log n\})$ and hence the total runtime of the algorithm is $O(\sqrt{kn}\cdot n/k \cdot \log k \min\{k, \log n\} \log(1/\epsilon))$ which is $O(n^{3/2} \log(1/\epsilon))$ for k constant and $O(n^{3/2}/k^{1/2} \cdot \log^2 n \cdot \log(1/\epsilon))$ for general k. The claim follows using $k \in \Omega(S/\log n)$. ■

3. Sliding Windows

Let D and R be two finite sets and $f : D^n \to R$ be a function over strings of length n. We define the operation $\boxplus$ which takes f and returns a function $f^{\boxplus t} : D^{n+t-1} \to R^t$, defined by $f^{\boxplus t}(x) = (f(x_i \ldots x_{i+n-1}))_{i=1}^t$.

3.1. Element Distinctness over Sliding Windows

The main result of this section shows that our randomized branching program for ED_n can even be extended to a $T \in \tilde{O}(n^{3/2}/S^{1/2})$ randomized branching program for its sliding windows version $ED_n^{\boxplus n}$. We do this in two steps. We first give a deterministic reduction which shows how the answer to an element distinctness problem allows one to reduce the input size of sliding-window algorithms for computing $ED_n^{\boxplus m}$.

Lemma 3.1. *Let $n > m > 0$.*
(a) If $ED_{n-m+1}(x_m, \ldots, x_n) = 0$ then
$ED_n^{\boxplus m}(x_1, \ldots, x_{n+m-1}) = 0^m$.
(b) If $ED_{n-m+1}(x_m, \ldots, x_n) = 1$ then define
$i_L = \max\{j \in [m{-}1] \mid ED_{n-j+1}(x_j, \ldots, x_n) = 0\}$
where $i_L = 0$ if the set is empty and define
$i_R = \min\{j \in [m{-}1] \mid ED_{n-m+j}(x_m, \ldots, x_{n+j}) = 0\}$
where $i_R = m$ if the set is empty. Then

$$ED_n^{\boxplus m}(x_1, \ldots, x_{n+m-1}) = 0^{i_L}1^{m-i_L} \;\wedge\; 1^{i_R}0^{m-i_R} \;\wedge\; ED_{m-1}^{\boxplus m}(x_1, \ldots, x_{m-1}, x_{n+1}, \ldots, x_{n+m-1})$$

where each $\wedge$ represents bit-wise conjunction.

Proof: The elements $M = (x_m, \ldots, x_n)$ appear in all m of the windows so if this sequence contains duplicated elements, so do all of the windows and hence the output for all windows is 0. This implies part (a).

If M does not contain any duplicates then any duplicate in a window must involve at least one element from

$L = (x_1, \ldots, x_{m-1})$ or from $R = (x_{n+1}, \ldots, x_{n+m-1})$. If a window has value 0 because it contains an element of L that also appears in M, it must also contain the rightmost such element of L and hence any window that is distinct must begin to the right of this rightmost such element of L. Similarly, if a window has value 0 because it contains an element of R that also appears in M, any window that is distinct must end to the left of the leftmost such element of R. The only remaining duplicates that can occur in a window can only involve elements of both L and R. In order, the m windows contain the following sequences of elements of $L \cup R$: $(x_1, \ldots, x_{m-1})$, $(x_2, \ldots, x_{m-1}, x_{n+1})$, ..., $(x_{m-1}, x_{n+1}, \ldots, x_{n+m-2})$, $(x_{n+1}, \ldots, x_{n+m-1})$. These are precisely the sequences for which $ED_{m-1}^{\boxplus m}(x_1, \ldots, x_{m-1}, x_{n+1}, \ldots, x_{n+m-1})$ determines distinctness. Hence part (b) follows. ■

We use the above reduction in input size to show that any efficient algorithm for element distinctness can be extended to solve element distinctness over sliding windows at a small additional cost.

Lemma 3.2. *If there is an algorithm A that solve element distinctness, ED, using time at most $T(n)$ and space at most $S(n)$, where T and S are nondecreasing functions of n, then there is an algorithm A^* that solves the sliding-window version of element distinctness, $ED_n^{\boxplus n}$, in time $T^*(n)$ that is $O(T(n) \log^2 n)$ and space $S^*(n)$ that is $O(S(n) + \log^2 n)$. Moreover, if $T(n)$ is $\Omega(n^\beta)$for $\beta > 1$, then $T^*(n)$ is $O(T(n) \log n)$.*

If A is deterministic then so is A^*. If A is randomized with error at most ϵ then A^* is randomized with error $o(1/n)$. Moreover, if A has 1-sided error (only false positives) then the same property holds for A^*.

Proof: We first assume that A is deterministic. Algorithm A^* will compute the n outputs of $ED_n^{\boxplus n}$ in n/m groups of m using the input size reduction method from Lemma 3.1. In particular, for each group A^* will first call A on the middle section of input size $n - m + 1$ and output 0^m if A returns 0. Otherwise, A^* will do two binary searches involving at most $2 \log m$ calls to A on inputs of size at most n to compute i_L and i_R as defined in part (b) of that lemma. Finally, in each group, A^* will make one recursive call to A^* on a problem of size m.

It is easy to see that this yields a recurrence of the form

$$T^*(n) = (n/m)[cT(n) \log m + T^*(m)].$$

In particular, if we choose $m = n/2$ then we obtain $T^*(n) \leq 2T^*(n/2) + 2cT(n) \log n$. If $T(n)$ is $\Omega(n^\beta)$ for $\beta > 1$ this solves to $T^*(n) \in O(T(n) \log n)$. Otherwise, it is immediate from the definition of $T(n)$ that $T(n)$ must be $\Omega(n)$ and hence the recursion for A^* has $O(\log n)$ levels and the total cost associated with each of the levels of the recursion is $O(T(n) \log n)$. The space for all the calls to A can be re-used in the recursion. and the algorithm A^* only needs to remember a constant number of pointers for each level of recursion for a total cost of $O(\log^2 n)$ additional bits.

We now suppose that the algorithm A is randomized with error at most ϵ. For the recursion based on Lemma 3.1, we use algorithm A and run it $C = O(\log n)$ times on input $(x_m, \ldots, x_n)$, taking the majority of the answers to reduce the error to $o(1/n^2)$. In case that no duplicate is found in these calls, we then apply the noisy binary search method of [20] to determine i_L and i_R with error at most $o(1/n^2)$ by using only $C = O(\log n)$ calls to A. (If the original problem size is n we will use the same fixed number $C = O(\log n)$ of calls to A even at deeper levels of the recursion so that each subproblem has error $o(1/n^2)$.) There are only $O(n)$ subproblems so the final error is $o(1/n)$. The rest of the run-time analysis is the same as in the deterministic case.

If A has only has false positives (if it claims that the input is not distinct then it is certain that there is a duplicate) then observe that A^* will only have false positives. ■

Combining Theorem 2.4 with Lemma 3.2 we obtain our algorithm for element distinctness over sliding windows.

Theorem 3.3. *For space $S \in [c \log n, n]$, $ED^{\boxplus n}$ can be solved in time $T \in O(n^{3/2} \log^{7/2} n / S^{1/2})$ with 1-sided error probability $o(1/n)$. If the space $S \in O(\log n)$ then the time is reduced to $T \in O\big(n^{3/2} \log n\big)$.*

When the input alphabet is chosen uniformly at random from $[n]$ there exists a much simpler 0-error sliding-window algorithm for $ED^{\boxplus n}$ that is efficient on average.

Theorem 3.4. *For input randomly chosen uniformly from $[n]^{2n-1}$, $ED^{\boxplus n}$ can be solved in average time $\overline{T} \in O(n)$ and average space $\overline{S} \in O(\log n)$.*

By way of contrast, under the same distribution, we prove an average case time-space lower bound of $\overline{T} \in \Omega(n^2/\overline{S})$ for $(F_0 \bmod 2)^{\boxplus n}$ in the next section.

3.2. Frequency Moments over Sliding Windows

We now show a $T \in \Omega(n^2/S)$ lower bound for randomized branching programs computing frequency moments over sliding windows. This contrasts with our significantly smaller $T \in \tilde{O}(n^{3/2}/S^{1/2})$ upper bound from the previous section for computing element distinctness over sliding windows in this same model, hence separating the complexity of ED and F_k for $k \neq 1$ over sliding windows. Our lower bound also applies to $F_0 \bmod 2$.

3.2.1. A general sequential lower bound for $F_k^{\boxplus n}$ and $(F_0 \bmod 2)^{\boxplus n}$: We derive a time-space tradeoff lower bound for randomized branching programs computing $F_k^{\boxplus n}$ for $k = 0$ and $k \geq 2$. Further, we show that the lower bound also holds for computing $(F_0 \bmod 2)^{\boxplus n}$. (Note that the parity of F_k for $k \geq 1$ is exactly equal to the parity of n; thus the outputs of $(F_k \bmod 2)^{\boxplus n}$ are all equal to $n \bmod 2$.)

Theorem 3.5. *Let $k = 0$ or $k \geq 2$. There is a constant $\delta > 0$ such that any $[n]$-way branching program of time T and space S that computes $F_k^{\boxplus n}$ with error at most η, $0 < \eta < 1 - 2^{-\delta S}$, for input randomly chosen uniformly from $[n]^{2n-1}$ must have $T \cdot S \in \Omega(n^2)$. The same lower bound holds for $(F_0 \bmod 2)^{\boxplus n}$.*

Corollary 3.6. *Let $k = 0$ or $k \geq 2$. (a) The average time $\overline{T}$ and average space $\overline{S}$ needed to compute $(F_k)^{\boxplus n}(x)$ for x randomly chosen uniformly from $[n]^{2n-1}$ satisfy $\overline{T} \cdot \overline{S} \in \Omega(n^2)$. (b) For $0 < \eta < 1 - 2^{-\delta S}$, any η-error randomized RAM or word-RAM algorithm computing $F_k^{\boxplus n}$ using time T and space S satisfies $T \cdot S \in \Omega(n^2)$.*

Proof of Theorem 3.5: We present the lower bound for $F_0^{\boxplus n}$; the relatively straightforward modifications for $k \geq 2$ and for computing $(F_0 \bmod 2)^{\boxplus n}$ are given in the full paper. For convenience, on input $x \in [n]^{2n-1}$, we write y_i for the output $F_k(x_i, \ldots, x_{i+n-1})$.

We use the general approach of Borodin and Cook [12] together with the observation of [3] of how it applies to average case complexity and randomized branching programs: We assume w.l.o.g. that branching program B of length T is leveled and divide it into layers of height q each. Each layer is now a collection of small branching programs B', each of whose start nodes is a node at the top level of that layer. Since the branching program must produce n outputs for each input x, for every input x there exists a small branching program B' of height q in some layer that produces at least $nq/T > S$ outputs. There are at most 2^S nodes in B and hence there are at most 2^S such small branching programs among all the layers of B. One would normally prove that the fraction of $x \in [n]^{2n-1}$ for which any one such small program correctly produces nq/T outputs is much smaller than 2^{-S} which implies the desired lower bound.

This outline is more complicated in our argument: If a small program B' finds that certain values are equal, then the answers to nearby windows may be strongly correlated (e.g., if $x_i{=}x_{i+n}$ then $y_i{=}y_{i+1}$) which may make correct outputs too likely. Therefore, we only reason about the outputs from positions that are not duplicated in the input. Moreover, inputs for which the value of F_0 in a window is extreme, say n (all distinct) or 1 (all identical), allow an almost-certain prediction of the value of F_0 for the next window. We show that with high probability under the uniform distribution there will be a linear number of unduplicated input positions and extreme values of F_0 do not occur.

The following is an easy application of Azuma-Hoeffding bounds and the observation that the expected number of distinct elements approaches $(1 - 1/e)n$.

Lemma 3.7. *Let a be chosen uniformly at random from $[n]^n$. Then the probability that $F_0(a)$ is between $0.5n$ and $0.85n$ is at least $1 - 2e^{-n/50}$.*

We say that x_j is *unique in* x if and only if $x_j \notin \{x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_{2n-1}\}$.

Lemma 3.8. *Let x be chosen uniformly at random from $[n]^{2n-1}$ with $n \geq 2$. With probability at least $1 - 4ne^{-n/50}$, (a) all outputs of $F_0^{\boxplus n}(x)$ are between $0.5n$ and $0.85n$, and (b) the number of positions $j < n$ such that x_j is unique in x is at least $n/24$.*

Correctness of small branching programs:

DEFINITION 3.1. *Let B' be an $[n]$-way branching program and let π be a source-sink path in B' with queries Q_π and answers $A_\pi : Q_\pi \to [n]$. An index $\ell < n$ is said to be π-*unique *iff either (a) $\ell \notin Q_\pi$, or (b) $A_\pi(\ell) \notin A_\pi(Q_\pi - \{\ell\})$.*

To measure the correctness of a small branching program, we restrict our attention to outputs that are produced at positions that are π-unique and upper-bound the probability that a small branching program correctly computes outputs of $F_0^{\boxplus n}$ at many π-unique positions in the input.

Let $\mathcal{E}$ be the event that all outputs of $F_0^{\boxplus n}(x)$ are between $0.5n$ and $0.85n$.

Lemma 3.9. *Let $r > 0$ be a positive integer, let $\epsilon \leq 1/10$, and let B' be an $[n]$-way branching program of height $q = \epsilon n$. Let π be a path in B' on which outputs from at least r π-unique positions are produced. For random x uniformly chosen from $[n]^{2n-1}$, conditioned on $\pi_{B'}(x) = \pi$, the probability that these r outputs are correct for $F_0^{\boxplus n}(x)$ and the event $\mathcal{E}$ also holds is at most $(17/18)^r$.*

Proof: Roughly, we will show that when $\mathcal{E}$ holds (outputs for all windows are not extreme) then, conditioned on following any path π in B', each output produced for a π-unique position will have only a constant probability of success conditioned on any outcome for the previous outputs. Because of the way outputs are indexed, it will be convenient to consider these outputs in right-to-left order.

Let π be a path in B', Q_π be the set of queries along π, $A_\pi : Q_\pi {\to} [n]$ be the answers along π, and $Z_\pi : [n]{\to}[n]$ be the partial function denoting the outputs produced along π. Note that $\pi_{B'}(x){=}\pi$ if and only if $x_i{=}A_\pi(i)$ for all $i{\in}Q_\pi$.

Let $1{\leq}i_1{<}\ldots{<}i_r{<}n$ be the first r of the π-unique positions on which π produces output values; i.e., $\{i_1, \ldots, i_r\} \subseteq \mathrm{dom}(Z_\pi)$. Define $z_{i_1}{=}Z_\pi(i_1), \ldots, z_{i_r}{=}Z_\pi(i_r)$.

We will decompose the probability over the input x that $\mathcal{E}$ and all of $y_{i_1}{=}z_{i_1}, \ldots, y_{i_r}{=}z_{i_r}$ hold via the chain rule. In order to do so, for $\ell \in [r]$, we define event $\mathcal{G}_\ell$ to be that $y_{i_\ell}{=}z_{i_\ell}, \ldots, y_{i_r}{=}z_{i_r}$ and event $\mathcal{E}_\ell$ to be $0.5n \leq F_0^{(i)}(x) \leq 0.85n$ for all $i > i_\ell$. We also write $\mathcal{E}_0 \stackrel{\text{def}}{=} \mathcal{E}$. Then

$$\begin{aligned}&\Pr[y_{i_1}{=}z_{i_1}, \ldots, y_{i_r}{=}z_{i_r},\ \mathcal{E} \mid \pi_{B'}(x){=}\pi]\\ &= \Pr[\mathcal{E}_r \mid \pi_{B'}(x){=}\pi]\\ &\quad \times \prod\nolimits_{\ell=1}^{r} \Pr[y_{i_\ell}{=}z_{i_\ell},\ \mathcal{E}_{\ell-1} \mid \mathcal{G}_{\ell+1},\ \mathcal{E}_\ell,\ \pi_{B'}(x){=}\pi]\end{aligned}$$

$$\leq \prod_{\ell=1}^{r} \Pr[y_{i_\ell}{=}z_{i_\ell} \mid \mathcal{G}_{\ell+1},\ \mathcal{E}_\ell,\ \pi_{B'}(x){=}\pi]. \tag{1}$$

We now upper bound each term in the product in (1). Depending on how much larger $i_{\ell+1}$ is than i_ℓ, the conditioning on the value of $y_{i_{\ell+1}}$ may imply a lot of information about the value of y_{i_ℓ}, but we will show that even if we reveal more about the input, the value of y_{i_ℓ} will still have a constant amount of uncertainty.

For $i \in [n]$, let W_i denote the vector of input elements $(x_i, \ldots, x_{i+n-1})$, and note that $y_i = F_0(W_i)$; we call W_i the i^{th} window of x. The values y_i for different windows may be closely related. In particular, adjacent windows W_i and W_{i+1} have numbers of distinct elements that can differ by at most 1 and this depends on whether the extreme end-points of the two windows, x_i and x_{i+n}, appear among their common elements $C_i = \{x_{i+1}, \ldots, x_{i+n-1}\}$. More precisely,

$$y_i - y_{i+1} = \mathbf{1}_{\{x_i \notin C_i\}} - \mathbf{1}_{\{x_{i+n} \notin C_i\}}. \tag{2}$$

In light of (2), the basic idea of our argument is that, because i_ℓ is π-unique and because of the conditioning on $\mathcal{E}_\ell$, there will be enough uncertainty about whether or not $x_{i_\ell} \in C_{i_\ell}$ to show that the value of y_{i_ℓ} is uncertain even if we reveal

1) the value of the indicator $\mathbf{1}_{\{x_{i_\ell+n} \notin C_{i_\ell}\}}$, and
2) the value of the output $y_{i_\ell+1}$.

We now make this idea precise by bounding each term in the product in (1). Set $\mathcal{H}_{\ell,B'} \overset{\text{def}}{=} (\mathcal{G}_{\ell+1} \wedge \mathcal{E}_\ell \wedge \pi_{B'}(x){=}\pi)$ and $\mathcal{Y}_{\ell,m,b} \overset{\text{def}}{=} (y_{i_\ell+1}{=}m \wedge \mathbf{1}_{\{x_{i_\ell+n} \notin C_{i_\ell}\}}{=}b)$.

$$\begin{aligned}
&\Pr[y_{i_\ell}{=}z_{i_\ell} \mid \mathcal{H}_{\ell,B'}] \\
&= \sum_{\substack{m\in[1,n]\\ b\in\{0,1\}}} \Pr[y_{i_\ell}{=}z_{i_\ell} \mid y_{i_\ell+1}{=}m, \mathbf{1}_{\{x_{i_\ell+n} \notin C_{i_\ell}\}}{=}b, \mathcal{H}_{\ell,B'}] \\
&\qquad \times \Pr[y_{i_\ell+1}{=}m, \mathbf{1}_{\{x_{i_\ell+n} \notin C_{i_\ell}\}}{=}b \mid \mathcal{H}_{\ell,B'}] \\
&\leq \max_{\substack{m\in[0.5n,0.85n]\\ b\in\{0,1\}}} \Pr[y_{i_\ell}{=}z_{i_\ell} \mid y_{i_\ell+1}{=}m, \mathbf{1}_{\{x_{i_\ell+n} \notin C_{i_\ell}\}}{=}b, \mathcal{H}_{\ell,B'}] \\
&= \max_{\substack{m\in[0.5n,0.85n]\\ b\in\{0,1\}}} \Pr[\mathbf{1}_{\{x_{i_\ell} \notin C_{i_\ell}\}}{=}z_{i_\ell}{-}m{+}b \mid \mathcal{Y}_{\ell,m,b}, \mathcal{H}_{\ell,B'}] \qquad (3)
\end{aligned}$$

where the inequality follows because the conditioning on $\mathcal{H}_{\ell,B'}$ and hence on $\mathcal{E}_\ell$ implies that $y_{i_\ell+1}$ is between $0.5n$ and $0.85n$ and the last equality follows because of the conditioning together with (2) applied with $i = i_\ell$. Obviously, unless $z_{i_\ell} - m + b \in \{0, 1\}$ the probability of the corresponding in the maximum in (3) will be 0. We will derive our bound by showing that given all the conditioning in (3), the probability of the event $\{x_{i_\ell} \notin C_{i_\ell}\}$ is between $2/5$ and $17/18$ and hence each term in the product in (1) is at most $17/18$.

Membership of x_{i_ℓ} in C_{i_ℓ}: First note that the condition $\mathcal{Y}_{\ell,m,b}$ (and hence $y_{i_\ell+1} = m$ and $\mathbf{1}_{\{x_{i_\ell+n} \notin C_{i_\ell}\}} = b$) implies that C_{i_ℓ} contains precisely $m - b$ distinct values. We now use the fact that i_ℓ is π-unique and, hence, either $i_\ell \notin Q_\pi$ or $A_\pi(i_\ell) \notin A_\pi(Q_\pi - \{i_\ell\})$.

First consider the case that $i_\ell \notin Q_\pi$ and the conditions $\mathcal{Y}_{\ell,m,b}$, $\mathcal{H}_{\ell,B'}$. By definition, the events $y_{i_\ell+1} = m$, $\mathbf{1}_{\{x_{i_\ell+n} \notin C_{i_\ell}\}} = b$, $\mathcal{E}_\ell$, and $\mathcal{G}_{\ell+1}$ only depend on x_i for $i > i_\ell$ and the conditioning on $\pi_{B'}(x) = \pi$ is only a property of x_i for $i \in Q_\pi$. Therefore, given conditions $\mathcal{Y}_{\ell,m,b}$ and $\mathcal{H}_{\ell,B'}$, x_{i_ℓ} is still a uniformly random value in $[n]$. Therefore the probability that $x_{i_\ell}{\in}C_{i_\ell}$ is precisely $(m{-}b)/n$ in this case.

Now assume that $i_\ell \in Q_\pi$. In this case, the conditioning on $\pi_{B'}(x) = \pi$ implies that $x_{i_\ell} = A_\pi(i_\ell)$ is fixed and not in $A_\pi(Q_\pi - \{i_\ell\})$. Again, from the conditioning we know that C_{i_ℓ} contains precisely $m - b$ distinct values. Some of the elements that occur in C_{i_ℓ} may be inferred from the conditioning – for example, their values may have been queried along π – but we will show that there is significant uncertainty about whether any of them equals $A_\pi(i_\ell)$. In this case we will show that the uncertainty persists even if we reveal (condition on) the locations of all occurrences of the elements $A_\pi(Q_\pi - \{i_\ell\})$ among the x_i for $i > i_\ell$.

Other than the information revealed about the occurrences of the elements $A_\pi(Q_\pi{-}\{i_\ell\})$ among the x_i for $i > i_\ell$, the conditioning on the events $y_{i_\ell+1}{=}m$, $\mathbf{1}_{\{x_{i_\ell+n} \notin C_{i_\ell}\}}{=}b$, $\mathcal{E}_\ell$, and $\mathcal{G}_{\ell+1}$, only biases the numbers of distinct elements and patterns of equality among inputs x_i for $i > i_\ell$. Further, the conditioning on $\pi_{B'(x)}{=}\pi$ does not reveal anything more about the inputs in C_{i_ℓ} than is given by the occurrences of $A_\pi(Q_\pi{-}\{i_\ell\})$.

Let $q' = |A_\pi(Q_\pi - \{i_\ell\})| \leq q - 1$ and let $q'' \leq q'$ be the number of distinct elements of $A_\pi(Q_\pi{-}\{i_\ell\})$ that appear in C_{i_ℓ}. Therefore, since the input is uniformly chosen, subject to the conditioning on $\mathcal{Y}_{\ell,m,b}$ and $\mathcal{H}_{\ell,B'}$, there are $m{-}b{-}q''$ distinct elements of C_{i_ℓ} not among $A_\pi(Q_\pi - \{i_\ell\})$, and these distinct elements are uniformly chosen from among the elements $[n]{-}A_\pi(Q_\pi{-}\{i_\ell\})$. Therefore, the probability that any of these $m - b - q''$ elements is equal to $x_{i_\ell} = A_\pi(i_\ell)$ is precisely $(m - b - q'')/(n - q')$ in this case.

It remains to analyze the extreme cases of the probabilities $(m - b)/n$ and $(m - b - q'')/(n - q')$ from the discussion above. Since $q = \epsilon n$, $q'' \leq q' \leq q - 1$, and $b \in \{0, 1\}$, we have the probability $\Pr[x_{i_\ell} \in C_{i_\ell} \mid \mathcal{Y}_{\ell,m,b}, \mathcal{H}_{\ell,B'}] \leq \frac{m}{n-q+1} \leq \frac{0.85n}{n-\epsilon n} \leq \frac{0.85n}{n(1-\epsilon)} \leq 0.85/(1 - \epsilon) \leq 17/18$ since $\epsilon \leq 1/10$. Similarly, $\Pr[x_{i_\ell} \notin C_{i_\ell} \mid \mathcal{Y}_{\ell,m,b}, \mathcal{H}_{\ell,B'}] < 1 - \frac{m-q}{n} \leq 1 - \frac{0.5n-\epsilon n}{n} \leq 0.5 + \epsilon \leq 3/5$ since $\epsilon \leq 1/10$. Plugging in the larger of these upper bounds in (1), we get: $\Pr[z_{i_1}, \ldots, z_{i_r}$ are correct for $F_0^{\boxplus n}(x),\ \mathcal{E} \mid \pi_{B'}(x) = \pi]$ is at most $(17/18)^r$, which proves the lemma. ■

Putting the pieces together: We now combine the above lemmas. Suppose that $TS \leq n^2/4800$ and let $q = n/10$. We can assume without loss of generality that $S \geq \log_2 n$ since we need $T \geq n$ to determine even a single answer.

Consider the fraction of inputs in $[n]^{2n-1}$ on which B correctly computes $F_0^{\boxplus n}$. By Lemma 3.8, for input x chosen uniformly from $[n]^{2n-1}$, the probability that $\mathcal{E}$ holds and there are at least $n/24$ positions $j < n$ such that x_j is unique in x is at least $1 - 4ne^{-n/50}$. Therefore, in order to be correct on any such x, B must correctly produce outputs from at least $n/24$ outputs at positions $j < n$ such that x_j is unique in x.

For every such input x, by our outline, one of the 2^S $[n]$-way branching programs B' of height q contained in B produces correct output values for $F_0^{\boxplus n}(x)$ in at least $r = (n/24)q/T \geq 20S$ positions $j < n$ such that x_j is unique in x.

We now note that for any B', if $\pi = \pi_{B'}(x)$ then the fact that x_j for $j < n$ is unique in x implies that j must be π-unique. Therefore, for all but a $4ne^{-n/50}$ fraction of inputs x on which B is correct, $\mathcal{E}$ holds for x and there is one of the $\leq 2^S$ branching programs B' in B of height q such that the path $\pi = \pi_{B'}(x)$ produces at least $20S$ outputs at π-unique positions that are correct for x.

Consider a single such program B'. By Lemma 3.9 for any path π in B', the fraction of inputs x such that $\pi_{B'}(x) = \pi$ for which $20S$ of these outputs are correct for x and produced at π-unique positions, and $\mathcal{E}$ holds for x is at most $(17/18)^{20S} < 3^{-S}$. By Proposition 3.8, this same bound applies to the fraction of all inputs x with $\pi_{B'}(x) = \pi$ for which $20S$ of these outputs are correct from x and produced at π-unique positions, and $\mathcal{E}$ holds for x is at most $(17/18)^{20S} < 3^{-S}$.

Since the inputs following different paths in B' are disjoint, the fraction of all inputs x for which $\mathcal{E}$ holds and which follow some path in B' that yields at least $20S$ correct answers from distinct runs of x is less than 3^{-S}. Since there are at most 2^S such height q branching programs, one of which must produce $20S$ correct outputs from distinct runs of x for every remaining input, in total only a $2^S 3^{-S} = (2/3)^S$ fraction of all inputs have these outputs correctly produced.

In particular this implies that B is correct on at most a $4ne^{-n/50} + (2/3)^S$ fraction of inputs. For n sufficiently large this is smaller than $1 - \eta$ for any $\eta < 1 - 2^{-\delta S}$ for some $\delta > 0$, which contradicts our original assumption. This completes the proof of Theorem 3.5. ■

3.2.2. A time-space efficient algorithm for $F_k^{\boxplus n}$: We now show that our time-space tradeoff lower bound for $F_k^{\boxplus n}$ is nearly optimal even for restricted RAM models.

Theorem 3.10. *There is a comparison-based deterministic RAM algorithm for computing $F_k^{\boxplus n}$ for any fixed integer $k \geq 0$ with time-space tradeoff $T \cdot S \in O(n^2 \log^2 n)$ for all space bounds S with $\log n \leq S \leq n$.*

Proof Sketch: Denote the i-th output of $F_k^{\boxplus n}$ by y_i. We first compute y_1 using the comparison-based $O(n^2/S)$ time sorting algorithm of Pagter and Rauhe [26]. This produces the outputs in order by building a space S data structure D and then repeatedly removing and returning (*popping*) the index of the smallest element from D. By keeping track of the latest symbol popped, we compute the frequency of each symbol and add its k-th power to the running total of F_k.

Let $S' = S/\log_2 n$. We compute the remaining outputs in n/S' groups of S' outputs at a time. In particular, suppose that we have already computed y_i. We compute $y_{i+1}, \ldots, y_{i+S'}$ by storing the values $x_i, \ldots, x_{i+S'-1}$ (the old elements) and $x_{i+n}, \ldots, x_{i+n+S'-1}$ (the new elements) in a binary search tree. We also keep a set of pointers associating each index of these elements with the leaf in the tree that represents it. For an old element, we keep a counter of its occurrences to its right in $x_i, \ldots, x_{i+S'-1}$ and similarly for a new one, we count its occurrences to its left in $x_{i+n}, \ldots, x_{i+n+S'-1}$. The $n - S'$ elements $x_{i+S'}, \ldots, x_{i+n-1}$ contribute to all the outputs, hence we scan them and maintain a counter at each leaf of the binary search tree to record the number of occurrences of a symbol in those common elements. Each symbol in $x_i, \ldots, x_{i+n+S'-1}$ has counters for its occurrences in the old elements, the new elements and the common elements $x_{i+S'}, \ldots, x_{i+n-1}$. For $j \in [i, i + S' - 1]$, we produce y_{j+1} from y_j by using the above counters to update the occurrences of x_j and x_{j+n} in the window $x_{j+1}, \ldots, x_{j+n}$.

The total storage required for the search trees and pointers is $O(S' \log n)$ which is $O(S)$. The total time to compute $y_{i+1}, \ldots, y_{i+S'}$ is $O(n \log S)$ time. This computation must be done $(n-1)/S$ times for a total of $O(\frac{n^2 \log S}{S})$ time. Hence, the total time including that to compute y_1 is $O(\frac{n^2 \log n \log S}{S})$ and hence $T \cdot S \in O(n^2 \log^2 n)$. ■

4. Order Statistics in Sliding Windows

When order statistics are extreme, their complexity over sliding windows does not significantly increase over that of a single instance.

Theorem 4.1. *There is a deterministic comparison algorithm that computes $MAX_n^{\boxplus n}$ (equivalently $MIN_n^{\boxplus n}$) using time $T \in O(n \log n)$ and space $S \in O(\log n)$.*

In contrast, when an order statistic is near the middle, we obtain a lower bound by a simple reduction using known time-space tradeoff lower bounds for sorting [12], [9].

Theorem 4.2. *For $t \in [n]$, any branching program computing $O_t^{\boxplus n}$ in time T and space S requires $T \cdot S \in \Omega(t^2)$. The same bound applies to expected time for randomized algorithms.*

For the median ($t = \lceil n/2 \rceil$), there is an errorless randomized algorithm for the single-input version with $T \in O(n \log \log_S n)$ for $S \in \omega(\log n)$ and this is tight for comparison algorithms [17]. This yields a significant separation in complexity between the sliding-window and single-input versions.

5. Discussion

Our algorithm for element distinctness can implemented by RAM algorithms using input randomness with the $T \in \tilde{O}(n^{3/2}/S^{1/2})$ bound. The impediment to implementing using online randomness is our use of truly random hash functions h. It seems plausible that a similar analysis would hold if those hash functions were replaced by some $\log^{O(1)} n$-wise independent hash function such as $h(x) = (p(x) \bmod m) \bmod n$ where p is a random polynomial of degree $\log^{O(1)} n$, which can be specified in space $\log^{O(1)} n$ and evaluated in time $\log^{O(1)} n$. This would suffice to yield essentially the same time-space tradeoff upper bound.

A time-space tradeoff separation between ED and F_k or $F_0 \bmod 2$ rather than their sliding windows versions remains an open question. In the context of quantum query complexity there is a separation between the complexities of the ED and $F_0 \bmod 2$: ED has quantum query complexity $\Theta(n^{2/3})$ ([1], [5]). On other hand, the lower bound in [10] implies that $F_0 \bmod 2$ has quantum query complexity $\Omega(n)$.

References

[1] S. Aaronson and Y. Shi, "Quantum lower bounds for the collision and the element distinctness problems," *J. ACM*, vol. 51(4), pp. 595–605, 2004.

[2] K. R. Abrahamson, "Generalized string matching," *SIAM Journal on Computing*, vol. 16(6), pp. 1039–1051, 1987.

[3] ——, "Time–space tradeoffs for algebraic problems on general sequential models," *Journal of Computer and System Sciences*, vol. 43(2), pp. 269–289, Oct. 1991.

[4] M. Ajtai, "A non-linear time lower bound for Boolean branching programs," *Theory of Computing*, vol. 1(1), pp. 149–176, 2005.

[5] A. Ambainis, "Quantum walk algorithm for element distinctness," *SIAM Journal on Computing*, vol. 37(1), pp. 210–239, 2007.

[6] A. Andersson and M. Thorup, "Dynamic ordered sets with exponential search trees," *J. ACM*, vol. 54(3), pp. 13:1–40, 2007.

[7] A. Arasu and G. S. Manku, "Approximate counts and quantiles over sliding windows," in *Proc. 23rd ACM PODS*, 2004, pp. 286–296.

[8] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems." in *Proc. 21st ACM PODS*, 2002, pp. 1–16.

[9] P. Beame, "A general sequential time-space tradeoff for finding unique elements," *SIAM Journal on Computing*, vol. 20(2), pp. 270–277, 1991.

[10] P. Beame and W. Machmouchi, "The quantum query complexity of AC^0," *Quantum Information & Computation*, vol. 12(7–8), pp. 670–676, 2012.

[11] P. Beame, M. Saks, X. Sun, and E. Vee, "Time-space tradeoff lower bounds for randomized computation of decision problems," *J. ACM*, vol. 50(2), pp. 154–195, 2003.

[12] A. Borodin and S. A. Cook, "A time-space tradeoff for sorting on a general sequential model of computation," *SIAM Journal on Computing*, vol. 11(2), pp. 287–297, May 1982.

[13] A. Borodin, F. E. Fich, F. Meyer auf der Heide, E. Upfal, and A. Wigderson, "A time-space tradeoff for element distinctness," *SIAM Journal on Computing*, vol. 16(1), pp. 97–99, Feb. 1987.

[14] V. Braverman, R. Ostrovsky, and C. Zaniolo, "Optimal sampling from sliding windows," *Journal of Computer and System Sciences*, vol. 78(1), pp. 260–272, 2012.

[15] R. P. Brent, "An improved Monte Carlo factorization algorithm," *BIT Numerical Mathematics*, vol. 20, pp. 176–184, 1980.

[16] A. Chakrabarti, G. Cormode, and A. McGregor, "A near-optimal algorithm for computing the entropy of a stream," in *Proc. 18th ACM-SIAM SODA*, 2007, pp. 328–335.

[17] T. M. Chan, "Comparison-based time-space lower bounds for selection," *ACM Transactions on Algorithms*, vol. 6(2), pp. 26:1–16, 2010.

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

[19] D. P. Dubhashi and A. Panconesi, *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2012.

[20] U. Feige, P. Raghavan, D. Peleg, and E. Upfal, "Computing with noisy information," *SIAM Journal on Computing*, vol. 23(5), pp. 1001–1018, 1994.

[21] D. E. Knuth, *Seminumerical Algorithms*, ser. The Art of Computer Programming. Addison-Wesley, 1971, vol. 2.

[22] L. K. Lee and H. F. Ting, "Maintaining significant stream statistics over sliding windows," in *Proc. 17th ACM-SIAM SODA*, 2006, pp. 724–732.

[23] ——, "A simpler and more efficient deterministic scheme for finding frequent items over sliding windows," in *Proc. 25th ACM PODS*, 2006, pp. 290–297.

[24] Y. Mansour, N. Nisan, and P. Tiwari, "The computational complexity of universal hashing," *Theoretical Computer Science*, vol. 107, pp. 121–133, 1993.

[25] G. Nivasch, "Cycle detection using a stack," *Information Processing Letters*, vol. 90(3), pp. 135–140, 2004.

[26] J. Pagter and T. Rauhe, "Optimal time-space trade-offs for sorting," in *Proc. 39th IEEE FOCS*, 1998, pp. 264–268.

[27] J. M. Pollard, "A Monte Carlo method for factorization," *BIT Numerical Mathematics*, vol. 15(3), pp. 331–334, 1975.

[28] ——, "Monte Carlo methods for index computation $(\bmod\, p)$," *Mathematics of Computation*, vol. 32(143), pp. 918–924, 1978.

[29] M. Sauerhoff and P. Woelfel, "Time-space tradeoff lower bounds for integer multiplication and graphs of arithmetic functions," in *Proc. 35th ACM STOC*, 2003, pp. 186–195.

[30] R. Sedgewick, T. G. Szymanski, and A. C.-C. Yao, "The complexity of finding cycles in periodic functions," *SIAM Journal on Computing*, vol. 11(2), pp. 376–390, 1982.

[31] P. C. van Oorschot and M. J. Wiener, "Parallel collision search with cryptanalytic applications," *Journal of Cryptology*, vol. 12(1), pp. 1–28, 1999.

[32] A. C. Yao, "Near-optimal time-space tradeoff for element distinctness," in *Proc. 29th IEEE FOCS*, 1988, pp. 91–97.

[33] Y. Yesha, "Time-space tradeoffs for matrix multiplication and the discrete Fourier transform on any general sequential random-access computer," *Journal of Computer and System Sciences*, vol. 29, pp. 183–197, 1984.