Space-efficient Query Evaluation over Probabilistic Event Streams

Rajeev Alur
University of Pennsylvania

Yu Chen
University of Pennsylvania

Kishor Jothimurugan
University of Pennsylvania

Sanjeev Khanna
University of Pennsylvania

Abstract

Real-time decision making in IoT applications relies upon space-efficient evaluation of queries over streaming data. To model the uncertainty in the classification of data being processed, we consider the model of probabilistic strings — sequences of discrete probability distributions over a finite set of events, and initiate the study of space complexity of streaming computation for different classes of queries over such probabilistic strings.

We first consider the problem of computing the probability that a word, sampled from the distribution defined by the probabilistic string read so far, is accepted by a given deterministic finite automaton. We show that this regular pattern matching problem can be solved using space that is only poly-logarithmic in the string length (and polynomial in the size of the DFA) if we are allowed a multiplicative approximation error. Then we show how to generalize this result to quantitative queries specified by additive cost register automata — these are automata that map strings to numerical values using finite control and registers that get updated using linear transformations. Finally, we consider the case when updates in such an automaton involve tests, and in particular, when there is a counter variable that can be either incremented or decremented but decrements only apply when the counter value is non-zero. In this case, the desired answer depends on the probability distribution over the set of possible counter values that can range from 0 to n for a string of length n. Under a mild assumption, namely probabilities of the individual events are bounded away from 0 and 1, we show that there is an algorithm that can compute all n entries of this probability distribution vector to within additive $1/\text{poly}(n)$ error using space that is only $\tilde{O}(\sqrt{n})$. In establishing these results, we introduce several new technical ideas that may prove useful for designing space-efficient algorithms for other query models over probabilistic strings.

CCS Concepts: • **Theory of computation → Streaming models**; **Streaming, sublinear and near linear time algorithms**; Regular languages.

Keywords: Query processing over streams, Streaming algorithms, Probabilistic streams.

ACM Reference Format:
Rajeev Alur, Yu Chen, Kishor Jothimurugan, and Sanjeev Khanna. 2020. Space-efficient Query Evaluation over Probabilistic Event Streams. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '20), July 8–11, 2020, Saarbrücken, Germany*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3373718.3394747

1 Introduction

Real-time decision making in IoT applications such as smart buildings and wearable devices relies upon processing of large data streams in an efficient and incremental manner. In this context, the input to the decision making policy can be modeled as a stream of data values, and a theoretical understanding of the trade-offs among space complexity, time complexity, and accuracy of the desired computation in a streaming fashion can inform the design and implementation of the policy [1, 11]. An example of an application domain where space-efficient streaming algorithms already play a key role is network traffic engineering which requires detection of traffic anomalies in real time while maintaining minimal state within the network router [3, 12, 14]. Our work is motivated by the observation that in many IoT applications there is an inherent uncertainty in the individual data items being processed. In such a case, the input should ideally be modeled as a stream of probability distributions over data values, and we need an understanding of trade-offs between resource bounds and accuracy for streaming computations over such probabilistic streams (see [5, 6] for early work on this topic).

In this paper, we restrict our attention to data streams specified as *probabilistic strings* over a finite alphabet Σ, where each item is an independent discrete probability distribution over Σ. Given a query over strings, that is, a function f from Σ^* to a set of decision values, one can naturally associate a

LICS '20, July 8–11, 2020, Saarbrücken, Germany

ACM ISBN 978-1-4503-7104-9/20/07…$15.00
https://doi.org/10.1145/3373718.3394747

function $\tilde{f}$ that maps a probabilistic string to a distribution over the decision values. Depending on the application, we may be interested in computing, given an input probabilistic string, how the expected decision value compares with a threshold value, or more generally, a profile of the distribution itself that can be queried. Our goal is to study, for different classes of queries f, the space complexity of solving such computational problems over probabilistic strings in a streaming fashion.

The first class of queries we study is the class of *regular languages* since not only is this the simplest and theoretically well-understood class, but is also integrated in query languages for specifying streaming computations in practical applications [3, 9]. In this case, given a deterministic finite automaton $\mathcal{A}$ and a probabilistic string, the streaming algorithm is tasked with computing the probability that $\mathcal{A}$ accepts the string read so far at every step. A probabilistic string naturally defines a current probability distribution over the finite set of states of $\mathcal{A}$, but the space required to store the *exact* probabilities in this distribution can grow linearly with the length of the input. We show that standard geometric discretization technique used in approximation algorithms as well as in streaming algorithms can be adapted for this problem resulting in a single-pass streaming algorithm that computes the answer to within a desired multiplicative accuracy and uses space that is only poly-logarithmic in the size of the input string and polynomial in the size of the DFA $\mathcal{A}$ (Theorem 3.3).

Next we turn our attention to computation of quantitative queries over strings specified using *cost register automata* — these are automata that map strings to numerical values using a DFA-based control, which can be used to detect regular patterns in input streams, and a set of registers that are updated on transitions, which can be used to compute quantitative properties depending on which operations are allowed during updates [2]. In additive cost register automata, all updates have to be linear transformations, and in this case we show how probabilistic strings can be processed in a streaming fashion to compute the expected value of the output of the automaton within a multiplicative approximation error and space that is polylogarithmic in the size of the input string (Theorem 4.2).

New challenges arise when the specification of a query f mapping strings to numerical values involves *conditional* updates to registers. The simplest form of conditional updates correspond to the case when there is a counter variable that can either be incremented or decremented, but a decrement is applicable only when the counter value is non-zero. Behavior modeled by such a counter is common in quantitative queries that involve requests and corresponding responses. While such queries can be evaluated efficiently in a streaming fashion over strings, when individual events involve uncertainty, we want to know if we can estimate the counter value efficiently in a streaming fashion.

In this setting with conditional updates, given a probabilistic string of length n, the information relevant to making a decision now is a probability distribution over the counter values which can range from 0 to n. As such a complete specification of this probability distribution requires $\Omega(n)$ space, even ignoring the precision issues (the probabilities for certain counter values can be exponentially small). In fact, the task of computing this probability distribution in sublinear space is non-trivial even if there is a standalone counter, with no associated finite state control, that is undergoing probabilistic increments and decrements with a barrier at 0 that prevents the counter value from going below 0. Our first key technical contribution is to handle the non-linearity introduced by the barrier at 0. We show that using only $\tilde{O}(\sqrt{n})$ space, we can compute the $(n+1)$-dimensional vector of probabilities that gives for each $i \in \{0, 1, 2, ..., n\}$ the probability that the counter has value i at the end of the stream, to within a total variation distance of $1/\text{poly}(n)$. Note that this means in particular, each entry in the vector is computed to within an additive error of $1/\text{poly}(n)$. We show this result by establishing a dynamic concentration result that allows us to compress the probability vector – we show that at any time, with high probability, the counter value is within a band of size $\tilde{O}(\sqrt{n})$. This band though is not static but is dynamically shifting as the stream progresses. But we show that it is possible to accurately track this band at each step and then focus on computing only the entries inside the band.

We finally extend this result to the setting where there is an underlying DFA based control and each probabilistic event results in a probabilistic transition to another state in the DFA. This introduces a new technical challenge, namely, an update to the counter value now can depend on the current state of the DFA; the current state in turn depends on the previous trajectory of the DFA. This means that the probability of an increment or decrement at any step now has potentially long-range dependencies to the events previously encountered in the stream. We use the *coupling technique* used in the analysis of Markov chains (see, [10], for instance) to bound these long-range dependencies under a mild assumption, namely probabilities of the individual events are bounded away from 0 and 1. It is worth highlighting that we prove this result without making any assumptions on the structure of the underlying DFA which is allowed to be an arbitrary directed graph where states can exhibit arbitrary reachability structure including non-trivial periodic behaviors. Putting together these ideas, we show that there is a single-pass streaming algorithm that can compute the probability distribution vector for a register undergoing conditional updates with an arbitrary underlying DFA based control to within additive $1/\text{poly}(n)$ error using space that is only $\tilde{O}(\sqrt{n})$ (Theorem 4.14).

We believe our approach of dynamically compressing probability distributions under conditional updates as well

as techniques for handling long-range dependencies when string processing is based on a DFA, may prove useful for designing space-efficient algorithms for other query models over probabilistic strings.

Organization: The rest of the paper is organized as follows. In Section 2 we describe the computational model for analyzing (streaming) query evaluation over probabilistic strings. In Section 3 we study regular queries specified using DFAs, and Section 4 deals with quantitative extensions of finite automaton using registers with linear updates and using a counter with barrier.

Related Work

The probabilistic data stream model introduced by Jayram et al. [5] is an extension of the standard data stream model [1, 11] which models data uncertainty using (sub)probability distribution functions (*pdf*s) over a discrete set of data values with bounded support. A (probabilistic) data point in a probabilistic data stream is given by a small number of value-probability pairs. The total probability of the specified values might be less than 1 in which case there is a non zero probability that the data point is missing. The model of probabilistic strings introduced here, in spite of being similar to probabilistic data streams, exhibits two key differences. Firstly, we consider probability distributions over *events* rather than data values where the set of events is finite and indicate qualitative properties about the underlying data. For example, an event a could represent the data value being in an interval (l_a, r_a). Secondly, the number of possible events is considered to be small allowing us to summarize the uncertainty in the data succinctly and more importantly, in a way that captures enough information to process certain queries. There has been many developments in streaming algorithms over probabilistic data streams [4, 6, 7] which consider the problem of computing the expected value of specific aggregation functions such as `MIN`, `MAX` and `AVERAGE`. In contrast, our focus is on generic algorithms that work for certain *classes* of queries. Moreover, the techniques used in these papers are tailored to estimate the expectation of the given aggregate function directly. Some of our techniques can be applied in a more general setting to estimate the entire probability distribution over the range of the query for some classes of queries.

Closely related to our work is the work on event queries [13] which presents a query language to specify qualitative properties of a sequence of events. The authors provide algorithms to compute the probability that the (probabilistic) stream seen so far satisfies a property specified in their language. Their technique is based on the ability to maintain a distribution over the set of states of a finite automaton over the set of events. However, the issue of increasing precision when multiplying probabilities is not addressed. We provide a way to deal with this problem and give precise theoretical guarantees on space complexity for "regular" queries as well as other, more complex, *quantitative* query classes.

2 The Computational Model

In this section, we define the notion of probabilistic strings and look at the definition and interpretation of queries over such strings. We also review the streaming model of computation in our context.

2.1 Probabilistic Strings

Let Σ be a finite set of input symbols. A string w over Σ is a finite sequence of symbols from Σ. We denote the set of all strings over Σ by Σ^*. We use $\mathcal{D}(\Sigma)$ to represent the set of all distributions over Σ.

$$\mathcal{D}(\Sigma) = \left\{ \pi : \Sigma \to [0,1] \;\middle|\; \sum_{\sigma \in \Sigma} \pi(\sigma) = 1 \right\}$$

Definition 2.1. *A probabilistic string α of length n over Σ is a finite sequence of distributions $\alpha_1, ..., \alpha_n$ where each $\alpha_i \in \mathcal{D}(\Sigma)$. The set of all probabilistic strings over Σ is denoted by $\mathcal{D}(\Sigma)^*$.*

Given a probabilistic string α of length n, $\mathcal{D}_\alpha$ represents the distribution over strings of length n in which the symbol at i^{th} position is drawn from the distribution α_i independent of the other positions. Hence, the probability of a string $w = w_1, ..., w_n$ under $\mathcal{D}_\alpha$ is given by $\Pi_{i=1}^{n} \alpha_i(w_i)$.

Example 2.2. *Consider $\Sigma = \{a, b\}$. A probabilistic string α of length n over Σ can be described by a sequence of numbers $p_1, ..., p_n$ where each $p_i \in [0,1]$ denotes the probability that the i^{th} symbol is a, whereas the probability of b is given by $1 - p_i$. The probability of the string a^n under $\mathcal{D}_\alpha$ is given by $\Pi_{i=1}^{n} p_i$.*

A string query is a function $f : \Sigma^* \to D$ where the range D is a subset of the reals. We extend string queries to queries over probabilistic strings by taking expectation. Given a string query f, we can define a function $\tilde{f} : \mathcal{D}(\Sigma)^* \to \mathsf{conv}(D)$[1] that maps a probabilistic string α to the expected value of f w.r.t. $\mathcal{D}_\alpha$.

$$\tilde{f}(\alpha) = \mathbb{E}_{w \sim \mathcal{D}_\alpha}[f(w)] = \sum_{|w|=n} f(w) \prod_{i=1}^{n} \alpha_i(w_i)$$

In this paper, we are particularly interested in two kinds of queries:

- $D = \{0, 1\}$. In this case, the query can be thought of as a language $L_f \subseteq \Sigma^*$ defined by $L_f = \{w \mid f(w) = 1\}$. The expected value of f w.r.t. $\mathcal{D}_\alpha$ is the same as the probability that a string, randomly chosen according to $\mathcal{D}_\alpha$, belongs to the language L_f, i.e., $\tilde{f}(\alpha) = \Pr_{w \sim \mathcal{D}_\alpha}[w \in L_f]$.

[1] $\mathsf{conv}(D)$ denotes the convex hull of D.

Example 2.3. *Consider $\Sigma = \{a, b\}$. Let f be the indicator function of the language a^*. Then, given $\alpha \in \mathcal{D}(\Sigma)^*$ of length n, $\tilde{f}(\alpha) = \Pi_{i=1}^{n} \alpha_i(a)$.*

- $D = \mathbb{N}$. In this case, f could be computing some quantitative property of the given string. Let us look at two examples:

  Example 2.4. *$\Sigma = \{a, b\}$ and f maps a string w to the sum of the number of a's and twice the number of b's in w. Given a probabilistic string α of length n, $\tilde{f}(\alpha) = \sum_{i=1}^{n} \alpha_i(a) + 2\sum_{i=1}^{n} \alpha_i(b) = n + \sum_{i=1}^{n} \alpha_i(b)$.*

  Example 2.5. *$\Sigma = \{a, b\}$ and f works as follows: Initialize a counter to 0; read the input string letter-by-letter, incrementing the counter on reading a and decrementing on reading b if the counter is non-zero; output the final value of the counter. Here the input string is viewed as a sequence of updates to a counter that always stays non-negative. It is not straightforward to characterize $\tilde{f}$ in this case and is studied in Section 4.*

2.2 The Streaming Model

The streaming model of computation was formalized in the seminal work of Alon, Matias, and Szegedy [1] as a framework for processing large data sets using small space. The past two decades have seen many exciting developments on streaming algorithms; we refer an interested reader to this survey [11]. In the streaming model, the input is a stream of values $v_1, ..., v_n$ from a discrete domain V and the goal is to compute some function g in a single pass over the input sequence, using space that is sublinear in n and polylogarithmic in $|V|$.

Given an integer k, let us denote by $\|k\|$ the number of bits in the binary representation of k. The size of a rational number $p = k_1/k_2$ is defined to be $\|p\| = \max(\|k_1\|, \|k_2\|)$. We will say that the description complexity of a distribution $\pi \in \mathcal{D}(\Sigma)$ is t if the probability of any symbol under π has size at-most t. The set of distributions over Σ with description complexity t is denoted by $\mathcal{D}(\Sigma)_t$.

$$\mathcal{D}(\Sigma)_t = \left\{ \pi \in \mathcal{D}(\Sigma) \;\middle|\; \forall \sigma,\ \|\pi(\sigma)\| \leq t \right\}$$

A probabilistic string has description complexity t if it is a sequence of distributions from $\mathcal{D}(\Sigma)_t$. Given a query f and a probabilistic string $\alpha \in \mathcal{D}(\Sigma)_t^*$ of length n, we are interested in the problem of computing $\tilde{f}(\alpha)$ in the streaming model, i.e., we want to compute $\tilde{f}(\alpha)$ in a single pass over α using space that is sublinear in n and polynomial in $|\Sigma|$ and t. Moreover, in some cases, we are also interested in computing the distribution of f under $\mathcal{D}_\alpha$ with similar space complexity.

Consider the query in Example 2.3. The problem here reduces to computing a product of n numbers. If the streaming algorithm is to simply store the product of the numbers seen so far, and update the product on reading every number, the space used could be linear in n because the precision of the product grows linearly in the size of the stream. In general, it might not be feasible to output exact answers and hence, we resort to algorithms that approximate the answer $\tilde{f}$ to a specified accuracy.

Formally, for any $\varepsilon \in (0, 1)$, we say that an estimate v of $\tilde{f}$ is an *ε-approximation* if $v \in [(1-\varepsilon)\tilde{f}, (1+\varepsilon)\tilde{f}]$. Note that, for $\varepsilon \in (0, 1/2)$, if we have an estimate v of $\tilde{f}$ such that $v \in [(1-\varepsilon^2)\tilde{f}, \frac{\tilde{f}}{(1-\varepsilon^2)}]$, it implies that v is an ε-approximation. We call an estimate v of $\tilde{f}$ *ε-accurate* if $v \in [(1-\varepsilon)\tilde{f}, \frac{\tilde{f}}{(1-\varepsilon)}]$. In this paper, we give algorithms that compute ε-approximate estimates by first computing ε^2-accurate estimates.

3 Regular Languages

In this section, we consider {0,1}-valued queries that are characteristic functions of a regular language, in other words, queries f such that L_f is regular. Any such query can be represented by a finite automaton over the input alphabet Σ.

We denote by $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ a Deterministic Finite Automaton (DFA) over Σ where Q is a finite set of states, $\delta : Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of final states. Given a DFA $\mathcal{A}$ and a probabilistic string $\alpha \in \mathcal{D}(\Sigma)_t^*$ of length n, the problem is to compute $\Pr_{w \sim \mathcal{D}_\alpha}[w \in \mathcal{L}(\mathcal{A})]$ where $\mathcal{L}(\mathcal{A})$ is the language of the automaton $\mathcal{A}$.

Example 2.3 is an instance of this problem. The challenge presented in Section 2.2, namely storing the product of numbers seen so far, can be handled using a geometric grouping idea. It turns out that the solution can be extended to deal with the general problem of computing the probability that a random string (drawn from the distribution defined by an input probabilistic string) is accepted by a given DFA. This is discussed in Section 3.2.

3.1 A Streaming Algorithm to Estimate Product

We present here a streaming algorithm to estimate the product of n numbers to a specified accuracy. Given a stream of rational numbers $p_1, ..., p_n$ from the range $[0, 1]$ such that $\|p_i\| \leq t$ for all $i \in \{1, ..., n\}$, and an $\varepsilon \in (0, 1/2)$, the goal is to find a number v such that $v \in [(1-\varepsilon)P, P/(1-\varepsilon)]$ where $P = \Pi_{i=1}^{n} p_i$.

Algorithm. The starting point for our algorithm is a partition of the range [0,1] into buckets such that each bucket is parameterized by a small integer. Let $\varepsilon' = \varepsilon/2n$. The bucket corresponding to an integer x is given by $B_{\varepsilon'}(x) = ((1-\varepsilon')^{x+1}, (1-\varepsilon')^x]$. Instead of storing the exact product, we only store the (approximate) bucket it belongs to. The product of two numbers represented by buckets $B_{\varepsilon'}(x)$ and $B_{\varepsilon'}(y)$ can be approximated by the bucket $B_{\varepsilon'}(x+y)$. This gives us a streaming algorithm (Algorithm 3.1) that approximates P.

Algorithm 3.1: Estimating product of n numbers

Initialize $x \leftarrow 0$;
for $i = 1, ..., n$ **do**
  On reading p_i, find the integer y_i such that $p_i \in B_{\varepsilon'}(y_i)$;
  Update $x \leftarrow x + y_i$;
Return $v = (1-\varepsilon')^x$

We next analyze the accuracy and the space requirements of the algorithm.

Approximation Guarantee. We can easily show by induction on i that, in the i^{th} iteration, $\Pi_{j=1}^{i} p_j \leq (1-\varepsilon')^x \leq (1/(1-\varepsilon')^i)\Pi_{j=1}^{i} p_j$. Therefore, $P \leq v$ and $P \geq (1-\varepsilon')^n v$. Using the binomial expansion of $(1-(\varepsilon/2n))^n$, we can see that, for large enough n, $(1-(\varepsilon/2n))^n \geq 1-\varepsilon$. Therefore, we have the guarantee that $v \geq P \geq (1-\varepsilon)v$.

Space Complexity. Since $\|p_i\|$ is bounded above by t, $p_i \geq 1/2^t$ for all i (If any p_i is zero the algorithm returns zero). Therefore for any i,

$$(1-\varepsilon')^{y_i} \geq 1/2^t$$

This gives $(e^{-\varepsilon'})^{y_i} \geq (1-\varepsilon')^{y_i} \geq 1/2^t$ where we have used the fact that $1+z \leq e^z$ forall z. Substituting the value of ε' and taking log on both sides, we get $\varepsilon y_i/2n \leq t\ln(2)$ and hence $y_i \leq 4nt/\varepsilon$. Observing that $x = \sum_{i=1}^{n} y_i$, we have $x \leq 4n^2t/\varepsilon$. This proves that the space used for storing x is $O(\log(4n^2t/\varepsilon))$.

In each iteration, y_i is computed using binary search. To do this, we need a space efficient way to compare numbers of the form $(1-\varepsilon')^y$ with p_i. We make use of the following lemma.

Lemma 3.1. *Given two rational numbers $z > 0$ and $p > 0$ and an integer y, we can check if $z^y < p$ using $O(\|y\|(\|y\| + \log(\|z\|)) + \log(\|p\|))$ space.*

Proof. WLOG assume $y > 0$. To compute z^y, exponentiation is done by repeated squaring using a space-efficient algorithm for multiplication. Suppose z is an integer. Let $\text{bit}_j(z^k)$ denote the j^{th} bit of z^k. Given integers $z > 0$, $k > 0$ and $j \geq 0$, we can compute $\text{bit}_j(z^k)$ recursively as follows.

We know that multiplication of two integers can be done in log-space. To compute $\text{bit}_j(z^k)$, we use a log-space algorithm A to compute the product of $z^{\lfloor k/2 \rfloor}$ and $z^{\lfloor k/2 \rfloor}$ (also multiply the product with z if k is odd). Keep a counter for the number of bits output by A. Whenever an output bit is produced, increment the counter and if the value of the counter is j, output that bit, ignoring it otherwise. When the algorithm needs to read a specific bit j' of $z^{\lfloor k/2 \rfloor}$, recursively compute $\text{bit}_{j'}(z, \lfloor k/2 \rfloor)$. If z is a rational we can compute any particular bit of the numerator or the denominator of z^y using the above procedure. Hence we can compare z^y with p bit-by-bit after cross multiplying the numerators with the denominators.

The recursion depth of the above computation is $O(\log(y))$, i.e., $O(\|y\|)$. Let z_{num} and z_{den} denote the numerator and denominator if z respectively. Then the space used by a recursive call is $O(\log(\|z_{\max}^y\|))$ where $z_{\max} = \max(z_{\text{num}}, z_{\text{den}})$. But $\|z_{\max}^y\| \leq y(\|z\|+1)$ and therefore $\log(\|z_{\max}^y\|) \leq \|y\| + \log(\|z\|+1)$ giving us the required space bound (The $\log(\|p\|)$ term shows up because of the cross multiplication step). We ignore the space needed to store the values z and p in this lemma as it is a part of storage space (which is included in the space used by the streaming algorithm separately) and is not a part of the additional space used by this comparison procedure. □

After rounding ε to the nearest power of $1/2$ we can ensure that $\|\varepsilon\| = O(\lceil \log(1/\varepsilon) \rceil)$. Making use of the fact that $0 \leq y_i \leq 4nt/\varepsilon$ and Lemma 3.1, we can compute y_i using $O(\log(nt/\varepsilon)(\log(nt/\varepsilon) + \log(\log(n/\varepsilon))) + \log(t))$ space. Hence, the total space used by the streaming algorithm is $O(\log(nt/\varepsilon)^2 + t)$.

3.2 Finite Automata

We now present a streaming algorithm to estimate the probability that a given probabilistic string belongs to a given regular language. We are given a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$. On input $\alpha \in \mathcal{D}(\Sigma)_t^*$ of length n and $\varepsilon \in (0, 1/2)$, let $P = \Pr_{w \sim \mathcal{D}_\alpha}[w \in \mathcal{L}(\mathcal{A})]$. We seek to find a number v such that $v \in [(1-\varepsilon)P, P/(1-\varepsilon)]$.

Algorithm. We maintain, for every state $q \in Q$, the probability (under $\mathcal{D}_\alpha$) of reaching q after reading the first i symbols, denoted by $\mathsf{P}^i(q)$ and finally, output the sum of probabilities of the final states. This can be done in a streaming fashion as described in Algorithm 3.2.

Algorithm 3.2: Computing DFA acceptance probability

Initialize: For every $q \in Q \setminus \{q_0\}$, set $\mathsf{P}^0(q) = 0$ and set $\mathsf{P}^0(q_0) = 1$;
for $i = 1, ..., n$ **do**
  On reading α_i, Update the P array:

$$\mathsf{P}^i(q) = \sum_{\substack{q', \sigma \\ \delta(q', \sigma) = q}} \mathsf{P}^{i-1}(q')\alpha_i(\sigma)$$

Return $v = \sum_{q \in F} \mathsf{P}^n(q)$

We approximate the values in P using buckets. Define $\varepsilon' = \varepsilon/2nm$ where m is the number of transitions in $\mathcal{A}$ (i.e., $|Q||\Sigma|$). We represent each value in the array P using a bucket $B_{\varepsilon'}(x)$ for some integer $x \in \mathbb{N}$. In the update step, multiplication adds the exponents, i.e., $B_{\varepsilon'}(x) \times B_{\varepsilon'}(y) \approx B_{\varepsilon'}(x+y)$ (Instead of $\alpha_i(\sigma)$ use $(1-\varepsilon')^{y_i(\sigma)}$ where $y_i(\sigma)$ is the integer such that

$\alpha_i(\sigma) \in B_{\varepsilon'}(y_i(\sigma))$). To perform addition, we add the terms two at a time. To add $B_{\varepsilon'}(x_1)$ and $B_{\varepsilon'}(x_2)$ we search for the integer y such that $(1-\varepsilon')^{x_1} + (1-\varepsilon')^{x_2} \in B_{\varepsilon'}(y)$ which is the same as the largest y such that $(1-\varepsilon')^y \geq (1-\varepsilon')^{x_1} + (1-\varepsilon')^{x_2}$. If such a y is less than 0, we use 0 instead.

The analysis is similar to Section 3.1.

Approximation Guarantee. Let $\hat{\mathsf{P}}^i(q)$ denote the (approximate) value of $\mathsf{P}^i(q)$ as computed by our algorithm where the value corresponding to a bucket $B_{\varepsilon'}(x)$ is $(1-\varepsilon')^x$. We can prove by induction on i that $\hat{\mathsf{P}}^i(q) \geq \mathsf{P}^i(q) \geq (1-\varepsilon')^{im}\hat{\mathsf{P}}^i(q)$ for all $q \in Q$ (In each iteration, the product introduces a multiplicative error of atmost $(1-\varepsilon')$ and there are $m-1$ additions each of which introduces a multiplicative error of atmost $(1-\varepsilon')$). Then, for large enough n, $v \geq P \geq (1-(\varepsilon/2nm))^{nm} v \geq (1-\varepsilon)v$.

Space Complexity. Since the smallest non-zero value possible is $1/2^{nt}$, and the estimates are always larger than the actual values, we have that for any $i \in \{1, ..., n\}$ and $q \in Q$, $\hat{\mathsf{P}}^i(q) \geq 1/2^{nt}$. If $\hat{\mathsf{P}}^i(q) = (1-\varepsilon')^x$, then $(1-\varepsilon/2nm)^x \geq 1/2^{nt}$ giving us $x \leq 4n^2mt/\varepsilon$. Therefore the space needed to store the array $\hat{\mathsf{P}}$ is $O(|Q|\log(nmt/\varepsilon))$. To bound the space used by the computation at each step, we need the following lemma in addition to Lemma 3.1.

Lemma 3.2. *Given a rational number z with $0 < z < 1$ and two integers x_1 and x_2 with $\|x_j\| \leq t_x$ for $j \in \{1, 2\}$, we can compute the largest integer y such that $z^y \geq z^{x_1} + z^{x_2}$ using $O(t_x(t_x + \log(\|z\|)))$ space.*

Proof. We perform a binary search for such a y. The comparison is done using the same approach as in the proof of Lemma 3.1 with the help of a recursive algorithm to compute particular bits of a given power of z. The addition in the RHS can also be done in log space. The overall recursion depth is $O(t_x)$ and the space used by a recursive call is $O(t_x + \log(\|z\|))$ giving us the required space bound. □

As before we can ensure that $\|\varepsilon\| = O(\lceil\log(1/\varepsilon)\rceil)$. Lemma 3.1 and Lemma 3.2 imply that the computation at each step can be performed using $O(\log(nmt/\varepsilon)^2)$ space. Hence the total space used is $O(\log(nmt/\varepsilon)^2 + |Q|\log(nmt/\varepsilon) + m\log(m) + t|\Sigma|)$ where the last two terms are simply the space needed to store the input DFA and the probability vector at each step. We can obtain an ε-approximation by computing an ε^2-accurate estimate, giving us the following theorem.

Theorem 3.3. *Given a DFA $\mathcal{A}$ over the alphabet Σ with m transitions, a probabilistic string $\alpha \in \mathcal{D}(\Sigma)^*_t$ of length n and an $\varepsilon \in (0, 1/2)$, we can compute an ε-approximation of $\Pr_{w\sim\mathcal{D}_\alpha}[w \in \mathcal{L}(\mathcal{A})]$ in a single pass over α using $O(t|\Sigma| + \log(nmt/\varepsilon)^2 + m\log(nmt/\varepsilon))$ space.*

4 Quantitative Properties

In this section, we look at $\mathbb{N}$-valued queries $f : \Sigma^* \to \mathbb{N}$, that can be specified by automata based models. We first consider finite state machines with registers that are updated using linear functions. Later, we investigate a model with finite states and a counter with two kinds of updates: increment and decrement, but with the constraint that the decrement applies only when the counter is non-zero.

4.1 Additive Cost Register Automata

We now consider a model called Cost Register Automata (CRA) introduced in [2]. It consists of a finite state control along with a finite number of registers. The state transitions are independent of the register values and the registers are updated based on the previous state and the symbol read. We consider a special case of CRAs in which the registers store values from $\mathbb{N}$ and the updates are linear. We call such machines *Additive CRAs*.

Formally, an Additive Cost Register Automaton (ACRA) $\mathcal{A}$ over the alphabet Σ is a tuple $(Q, \Sigma, \delta, q_0, r, \mathbf{M}, \mathbf{u}, \mathbf{v}_0, \boldsymbol{\rho})$ where Q is a finite set of states, $\delta : Q \times \Sigma \to Q$ is the state transition function, $q_0 \in Q$ is the initial state, $r \in \mathbb{N}$ is the number of registers, $\mathbf{M} : Q \times \Sigma \to \mathbb{N}^{r^2}$ and $\mathbf{u} : Q \times \Sigma \to \mathbb{N}^r$ dictate the register update function $\mu_\mathcal{A} : Q \times \Sigma \times \mathbb{N}^r \to \mathbb{N}^r$ given by $\mu_\mathcal{A}(q, \sigma, \mathbf{v}) = \mathbf{M}(q, \sigma)\mathbf{v} + \mathbf{u}(q, \sigma)$ where $\mathbf{M}(q, \sigma)$ is viewed as an $r \times r$ matrix. $\mathbf{v}_0 \in \mathbb{N}^r$ is the vector of initial values of the registers and $\boldsymbol{\rho} : Q \to \mathbb{N}^r$ is the aggregation function that combines the register values to produce a natural number.

A configuration of $\mathcal{A}$ is a pair consisting of a state and a vector representing register values, $(q, \mathbf{v}) \in Q \times \mathbb{N}^r$. On an input string $w = w_1, ..., w_n$, a run of $\mathcal{A}$ on w is a sequence of configurations $(q_0, \mathbf{v}_0), (q_1, \mathbf{v}_1), ..., (q_n, \mathbf{v}_n)$ such that for all $i \in \{1, ..., n\}$, $q_i = \delta(q_{i-1}, w_i)$ and $\mathbf{v}_i = \mu_\mathcal{A}(q_{i-1}, w_i, \mathbf{v}_{i-1})$. The output of the automaton, denoted $f_\mathcal{A}(w)$ is given by $\boldsymbol{\rho}(q_n)^T\mathbf{v}_n$.

On input an ACRA A and a probabilistic string $\alpha \in \mathcal{D}(\Sigma)^*_t$ we are interested in computing $\tilde{f}_\mathcal{A}(\alpha) = \mathbb{E}_{w\sim\mathcal{D}_\alpha}[f_\mathcal{A}(w)]$ (in the streaming model). We'll assume that all values in $\mathbf{v}_0$, $\mathbf{M}$, $\mathbf{u}$ and $\boldsymbol{\rho}$ can be represented using t bits.

Algorithm. Let $\mathsf{P}^i(q) \in [0, 1]$ denote the probability (under $\mathcal{D}_\alpha$) of reaching state q after reading the first i symbols. We use $\mathbf{v}(w) \in \mathbb{N}^r$ to denote the vector of register values after reading the string w. Let $\mathbf{E}^i_c(q) \in \mathbb{R}^r$ denote the expected values of the registers after reading the first i symbols, given that the state reached is q, i.e, $\mathbf{E}^i_c(q) = \mathbb{E}_{w\sim\mathcal{D}_\alpha}[\mathbf{v}(w_{\leq i}) \mid \delta(q_0, w_{\leq i}) = q]$ where $w_{\leq i}$ denotes the prefix of w of length i. Define $\mathbf{E}^i(q) = \mathsf{P}^i(q)\mathbf{E}^i_c(q)$. Then,

$$\mathbf{E}^i(q) = \sum_{\substack{|w'|=i \\ \delta(q_0,w')=q}} \mathcal{D}_\alpha(w')\mathbf{v}(w') \tag{1}$$

where $\mathcal{D}_\alpha(w') = \Pi_{j=1}^{i} \alpha_j(w'_j)$ is the probability that a string w drawn from $\mathcal{D}_\alpha$ has w' as a prefix. The streaming algorithm stores two arrays P and $\mathbf{E}$ which contain $\mathsf{P}^i(q)$ and $\mathbf{E}^i(q)$ respectively (for all q). P can be updated in the same way as in Algorithm 3.2. The following lemma gives the update rule for $\mathbf{E}$. The proof is straightforward using Equation 1 and can be found in Appendix A.1.

Lemma 4.1. *For all $i > 0$,*

$$\mathbf{E}^i(q) = \sum_{\substack{q',\sigma \\ \delta(q',\sigma)=q}} \alpha_i(\sigma)\left[\mathbf{M}(q',\sigma)\mathbf{E}^{i-1}(q') + \mathsf{P}^{i-1}(q')\mathbf{u}(q',\sigma)\right] \quad (2)$$

Now, $\tilde{f}_{\mathcal{A}}(\alpha) = \mathbb{E}_{w\sim\mathcal{D}_\alpha}[f_{\mathcal{A}}(w)]$ which can be stated in terms of conditional expectations as $\sum_{q\in Q} \mathsf{P}^n(q)\mathbb{E}[f_{\mathcal{A}}(w) \mid \delta(q_0, w) = q]$. Note that if the last state is q, then $f_{\mathcal{A}}(w) = \boldsymbol{\rho}(q)^T\mathbf{v}(w)$. Using linearity of expectation, we get $\mathbb{E}[f_{\mathcal{A}}(w)] = \sum_{q\in Q} \mathsf{P}^n(q)\boldsymbol{\rho}(q)^T\mathbf{E}_c^n(q) = \sum_{q\in Q} \boldsymbol{\rho}(q)^T\mathbf{E}^n(q)$. This gives us the streaming algorithm presented in Algorithm 4.1.

Algorithm 4.1: Computing expected output of an ACRA

Initialize P: For every $q \in Q \setminus \{q_0\}$, set $\mathsf{P}^0(q) = 0$ and set $\mathsf{P}^0(q_0) = 1$;
Initialize $\mathbf{E}$: For every $q \in Q \setminus \{q_0\}$, set $\mathbf{E}^0(q) = \mathbf{0} \in \mathbb{R}^r$ and set $\mathbf{E}^0(q_0) = \mathbf{v}_0$;
for $i = 1, \ldots, n$ **do**
  On reading α_i, update the P array:

$$\mathsf{P}^i(q) = \sum_{\substack{q',\sigma \\ \delta(q',\sigma)=q}} \mathsf{P}^{i-1}(q')\alpha_i(\sigma).$$

  Update the $\mathbf{E}$ array according to Equation 2.
Return $\sum_{q\in Q} \boldsymbol{\rho}(q)^T\mathbf{E}^n(q)$

As before, we use buckets to approximate the values in both the arrays. We are also given an approximation factor ε as input. Let $\varepsilon' = \varepsilon/4nmr$ where m is the number of transitions in $\mathcal{A}$. We use buckets of the form $B_{\varepsilon'}(x)$ to store values. Here, the values can be greater than 1 and hence x can be negative. Addition and multiplication is performed on buckets in the same way as in Section 3.

Approximation Guarantee. Let $\hat{\mathsf{P}}^i(q)$ and $\hat{\mathbf{E}}^i(q)$ denote the approximations of $\mathsf{P}^i(q)$ and $\mathbf{E}^i(q)$ respectively. Since the operations on buckets always over-estimate, we have that $\hat{\mathsf{P}}^i(q) \geq \mathsf{P}^i(q)$, and $\hat{\mathsf{E}}^i(q)_k \geq \mathsf{E}^i(q)_k$, for all $q \in Q$, $i \in [n]$, and $k \in [r]$, where the subscript k indicates the k^{th} component of the vector. Since there are atmost $2mr$ operations involved in computing any particular entry in each iteration and each operation introduces a multiplicative error of atmost $(1-\varepsilon')$, we get $\mathsf{P}^i(q) \geq (1-\varepsilon')^{2imr}\hat{\mathsf{P}}^i(q)$ and $\mathsf{E}^i(q)_k \geq (1-\varepsilon')^{2imr}\hat{\mathsf{E}}^i(q)_k$ for all q, i and k. If v is the final result of the algorithm, $v \geq \tilde{f}_{\mathcal{A}}(\alpha) \geq (1-\varepsilon/4nmr)^{2nmr}v \geq (1-\varepsilon)v$ for large n.

Space Complexity. It is easy to see that all non-zero values (in P and $\mathbf{E}$) are bounded from below by $1/2^{nt}$. Since we always maintain an over-estimate, for any bucket $B_{\varepsilon'}(x)$ stored in memory, $(1-\varepsilon/4nmr)^x \geq 1/2^{nt}$ which implies $x \leq 8n^2mrt/\varepsilon$. Also, note that the values in P are bounded above by 1 and values in $\mathbf{E}$ are bounded by the largest value any register can take for any input word of length n, which can be atmost $(4r)^{nt}$ (by induction on n). From the approximation guarantee we get that $(1-\varepsilon')^{x+2nmr} \leq (4r)^{nt}$ which proves that $x \geq -4n^2mrt\log(4r)/\varepsilon - 2nmr$. Hence the space used by a single value is $O(\log(nmrt/\varepsilon))$. Thus the total space for storing the two arrays $\hat{\mathsf{P}}$ and $\hat{\mathbf{E}}$ is $O(|Q|r\log(nmrt/\varepsilon))$.

Using Lemmas 3.1 and 3.2, the computation in any iteration can be performed using $O(\log(nmrt/\varepsilon)^2)$ space. Taking into account the space needed to store the input automaton and the input probabilities at each step, we arrive at the following theorem.

Theorem 4.2. *Given an ACRA $\mathcal{A}$ over the alphabet Σ with m transitions and r registers, a probabilistic string $\alpha \in \mathcal{D}(\Sigma)_t^*$ of length n and an $\varepsilon \in (0, 1/2)$, we can compute an ε-approximate estimate of the expected output of $\mathcal{A}$ w.r.t. $\mathcal{D}_\alpha$, $\tilde{f}_{\mathcal{A}}(\alpha)$ in a single pass over α using $O(\log(nmrt/\varepsilon)^2 + mr\log(nmt/\varepsilon) + mr^2 + t|\Sigma|)$ space.*

4.2 Adding Non-linear Updates

We have thus far considered a model where the register updates are linear. We now introduce a setting where the updates exhibit a non-linear behavior. Specifically, we will consider the case of a single register that behaves as a counter that undergoes increments and decrements but its value can never go below 0. In other words, the counter has a barrier at 0. As we will see, several new technical ideas are needed to handle this non-linearity.

The query here is specified using a finite automaton with a non-negative counter which is a special kind of CRAs given by $\mathcal{A} = (Q, \Sigma, \delta, q_0, u, v_0)$ where Q, δ and q_0 are defined as before. There is a single counter X. The counter update function is given by $u : Q \times \Sigma \to \{+1, -1\}$; on reading σ at state q, $u(q, \sigma)$ is either an increment or a decrement and $v_0 \in \mathbb{N}$ is the initial value of the counter. On an input string $w = w_1, \ldots, w_n$, a run of $\mathcal{A}$ on w is a sequence $(q_0, v_0), \ldots, (q_n, v_n)$ where each $(q_i, v_i) \in Q \times \mathbb{N}$ and for all $i \in \{1, \ldots, n\}$, $q_i = \delta(q_{i-1}, w_i)$ and $v_i = \max(v_{i-1} + u(q_{i-1}, w_i), 0)$. We are interested in estimating the expected value of the final value of the counter v_n w.r.t. $\mathcal{D}_\alpha$ for an input probabilistic string $\alpha \in \mathcal{D}(\Sigma)_t^*$.

To start with, we first consider an automaton which has only a single state, say q_0. Specifically, the setting of the problem is as follows: the symbol set Σ has two elements, say a and b. $u(q_0, a) = 1$ and $u(q_0, b) = -1$, in other words, the counter X will increase by 1 when the automaton processes

an a and will decrease by 1 when a b is processed; however, if the counter value is already zero, then a decrement is not applied and the counter value remains 0.

For this single state case, we will give an algorithm that computes an approximation of the *complete distribution* of the counter values after reading a probabilistic string $\alpha \in \mathcal{D}(\Sigma)_t^*$ of length n, with $\tilde{O}(\sqrt{n})$ space, where $\tilde{O}$ hides logarithmic factors. We measure the quality of approximation by the total variation distance between the true distribution and the distribution output by the algorithm. The total variation distance of two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ over a finite set V is given by $\text{TVD}(\mathcal{D}_1, \mathcal{D}_2) = \frac{1}{2}\sum_{v\in V}|\mathcal{D}_1(v) - \mathcal{D}_2(v)|$ where $\mathcal{D}_j(v)$ is the probability of v under $\mathcal{D}_j$ for $j \in \{1, 2\}$. We can thus use the (approximate) output distribution to compute the expected value of the counter w.r.t. $\mathcal{D}_\alpha$. The rest of this (sub) section is devoted to proving the following theorem.

Theorem 4.3. *There is a single pass streaming algorithm that, given a probabilistic string $\alpha \in \mathcal{D}(\Sigma)_t^*$ of length n, computes a probability distribution over the counter values at the end of the stream whose total variation distance from the true distribution of the counter values is $O\left(\frac{1}{n^2}\right)$ and uses $O\left(\sqrt{n\log(n)^3}+t\right)$ space.*

We start by setting up some notation. Let p_i be the probability that the i^{th} symbol is a and $(1 - p_i)$ be the probability that the i^{th} symbol is b (same as in Example 2.2). Let $X_1, X_2, \ldots, X_n$ be random variables where the variable X_i takes value 1 with probability p_i, and value -1 with probability $(1 - p_i)$. We denote by $X_{\le i}$ the counter value at step i (with barrier at 0). Our approach is based on linking the behavior of the counter with a barrier at 0 with that of a counter with no barrier at 0. We will refer to this latter counter as the *unrestricted counter*. For $1 \le i \le n$, we define $X^U_{\le i} = \sum_{j=1}^{i} X_i$. Thus $X^U_{\le i}$ denotes the counter value after i items have been processed if there were no barrier at 0, and the counter was initialized to 0. Finally, let $L_i = \min_{j\le i}\{X^U_{\le j}\}$ and $L^E_i = \min_{j\le i}\{\mathbb{E}[X^U_{\le j}]\}$, denote respectively the least value achieved and the least expected value of the unrestricted counter in processing the first i items in the stream.

Now we give two crucial properties of the counter with a barrier. Lemma 4.4 describes a connection between the counter with barrier and the unrestricted counter. Using this connection, we then prove in Lemma 4.5 that the value of the counter with barrier remains concentrated in a narrow range at any point. We will prove our result in the more general setting where the counter starts with an arbitrary initial value v_0 (which is typically 0). This general version will be useful for a result in the next section.

Lemma 4.4. *Suppose the counter value is initially set to v_0. Then for any $1 \le i \le n$, $X_{\le i} = v_0 + X^U_{\le i} - \min\{0, v_0 + L_i\}$.*

Proof. We first prove that if $v_0 = 0$ then $X_{\le i} = X^U_{\le i} - L_i$. Suppose $j \in \{1, \ldots i\}$ is the last time before step i such that $L_i = X^U_{\le j}$. We prove that j is then the last time before step i such that $X_{\le j} = 0$. The lemma then follows from the identity, $X_{\le i} = \sum_{k=j+1}^{i} X_k = X^U_{\le i} - X^U_{\le j}$ which holds because the barrier is never touched (i.e. the counter never becomes 0) after step j.

If j is not the last time before i with $X_{\le j} = 0$, then there are two cases; either the last time is before j or the last time is after j.

- **Case 1:** If the last time is before j, then $X_{\le j} > 0$. Suppose the last time is k where $k < j$, then the barrier is not touched between steps $(k + 1)$ through j, which means $X_{\le j} = X^U_{\le j} - X^U_{\le k}$. So $X^U_{\le j} > X^U_{\le k}$, which is a contradiction since $L_i = X^U_{\le j}$.
- **Case 2:** If the last time is after j. Let k be the *first* time that $X_{\le k} = 0$ after time j. The barrier is then not touched between steps $(j+1)$ through k, which means $X_{\le k} - X_{\le j} = X^U_{\le k} - X^U_{\le j}$. Since $X_{\le k} = 0, X^U_{\le k} \le X^U_{\le j}$, this is a contradiction since j is the last time that $L_i = X^U_{\le j}$.

When $v_0 > 0$, if $L_i + v_0 \le 0$ then the counter has touched the barrier at least once. Using a similar argument as above, we get $X_{\le i} = X^U_{\le i} - L_i = v_0 + X^U_{\le i} - (v_0 + L_i)$. If $L_i + v_0 > 0$, then the counter has never touched the barrier, the value is the same as the counter without a barrier, which means $X_{\le i} = v_0 + X^U_{\le i}$. □

Lemma 4.5. *Suppose the counter value is initially set to v_0. Then with probability at least $1 - \frac{1}{n^2}$, for any $1 \le i \le n$, $|X_{\le i} - (v_0 + \mathbb{E}[X^U_{\le i}] - \min\{v_0 + L^E_i, 0\})| < 16\sqrt{n\log n}$.*

Proof. Let $Y_i = (X_i + 1)/2$ and $Y_{\le i} = \sum_{j=1}^{i} Y_j$, then $X^U_{\le i} = 2Y_{\le i} - i$. Since Y_is are independent 0-1 random variables, by the Chernoff bounds, $|Y_{\le i} - \mathbb{E}[Y_{\le i}]| < 4\sqrt{n\log n}$ with probability at least $1 - \frac{1}{n^3}$, which means $|X^U_{\le i} - \mathbb{E}[X^U_{\le i}]| < 8\sqrt{n\log n}$ also with probability at least $1 - \frac{1}{n^3}$.

Applying union bound on all i, with probability at least $1 - \frac{1}{n^2}$, $|X^U_{\le i} - \mathbb{E}[X^U_{\le i}]| < 8\sqrt{n\log n}$ for every $1 \le i \le n$. In this event, for any i,

$$L_i = \min_{j\le i}\{X^U_{\le j}\} > \min_{j\le i}\{\mathbb{E}[X^U_{\le j}] - 8\sqrt{n\log n}\} = L^E_i - 8\sqrt{n\log n}$$

On the other hand, suppose j is a time when $L^E_i = \mathbb{E}[X^U_{\le j}]$, then

$$L_i \le X^U_{\le j} < \mathbb{E}[X^U_{\le j}] + 8\sqrt{n\log n} = L^E_i + 8\sqrt{n\log n}$$

So $|L_i - L^E_i| < 8\sqrt{n\log n}$ in this event. So for all $1 \le i \le n$, $|\min\{v_0 + L_i, 0\} - \min\{v_0 + L^E_i, 0\}| < 8\sqrt{n\log n}$.

By Lemma 4.4, $X_{\le i} = v_0 + X^U_{\le i} - \min\{0, v_0 + L_i\}$, which implies $|X_{\le i} - (v_0 + \mathbb{E}[X^U_{\le i}] - \min\{v_0 + L^E_i, 0\})| < 16\sqrt{n\log n}$ with probability at least $1 - \frac{1}{n^2}$. □

We next give an algorithm to compute the probability distribution of the counter values. For clarity of exposition, we first assume that we can store a probability value to arbitrary precision using only 1 bit of space. We then show how to eliminate this assumption.

Algorithm. For each $i \in \{1, ..., n\}$ and $v \in \mathbb{N} \cap [v_0 - n, v_0 + n]$, let $\mathsf{P}^i[v]$ denote the probability that the counter value is v after step i, i.e., the probability that $X_{\le i} = v$. We maintain an approximation of the array P^i which we denote by $\hat{\mathsf{P}}^i$. The key point is that we do not explicitly store $\mathsf{P}^i[v]$ for every possible counter value v but only for $O(\sqrt{n \log n})$ values of v. The values v of interest at step i (for $\hat{\mathsf{P}}^i$) will be precisely the integer values within a distance of $16\sqrt{n \log n}$ from $(v_0 + \mathbb{E}[X^U_{\le i}] - \min\{v_0 + L^E_i, 0\})$. If an entry v falls outside this range, we implicitly have $\hat{\mathsf{P}}^i[v] = 0$.

We now describe how the array $\hat{\mathsf{P}}^i$ is computed from the array $\hat{\mathsf{P}}^{i-1}$. It is easy to see that the values $\mathsf{P}^i[v]$ satisfy the following equation:

$$\mathsf{P}^i[v] = \begin{cases} p_i \cdot \mathsf{P}^{i-1}[v-1] + (1-p_i) \cdot \mathsf{P}^{i-1}[v+1] & \text{if } v > 0 \\ (1-p_i)(P^{i-1}[0] + P^{i-1}[1]) & \text{if } v = 0 \end{cases}$$

We use this equation to update $\hat{\mathsf{P}}^i$ storing only the values in the range of interest. Values not present in $\hat{\mathsf{P}}^{i-1}$ are treated as 0 when computing $\hat{\mathsf{P}}^i$. We can delete $\hat{\mathsf{P}}^{i-1}$ after computing $\hat{\mathsf{P}}^i$, so the total space used in storing the array $\hat{\mathsf{P}}^i$ after any step i is $O(\sqrt{n \log n})$. (Note that both $\mathbb{E}[X^U_{\le i}]$ and L^E_i can be maintained using only $O(\log n)$ space).

Lemma 4.6. $2\,\mathrm{TVD}(\hat{\mathsf{P}}^n, \mathsf{P}^n) = \sum_{v=v_0-n}^{v_0+n} |\hat{\mathsf{P}}^n[v] - \mathsf{P}^n[v]| < \frac{1}{n^2}$.

Proof. A realization $R = (R_0, R_1, R_2, \ldots, R_n)$ is a sequence of counter values such that for all $1 \le i \le n$, $X_{\le i} = R_i$ and $X_{\le 0} = v_0$. We say a realization R is *good until time i* if for each $j \le i$, $|R_j - (v_0 + \mathbb{E}[X^U_{\le j}] - \min\{v_0 + L^E_j, 0\})| < 16\sqrt{n \log n}$. We say a realization R is *bad* if R is not good until time n. Let $G_{i,v}$ be the set of all realizations R that are good until time i and $R_i = v$. We will prove that for any i, v, $\hat{\mathsf{P}}^i[v] = \sum_{R \in G_{i,v}} \Pr(R)$ where $\Pr(R)$ is the probability of R under $\mathcal{D}_\alpha$. With this equation, the lemma follows from Lemma 4.5 which shows that the total probability mass of bad realizations is bounded by $\frac{1}{n^2}$.

We prove $\hat{\mathsf{P}}^i[v] = \sum_{R \in G_{i,v}} \Pr(R)$ by induction on i. In the base case, if $i = 0$, since any possible realization R is good until i and has $R_0 = v_0$, $\sum_{R \in G_{i,v}} \Pr(R)$ is 1 if $v = v_0$ and 0 otherwise which is exactly the same as $\hat{\mathsf{P}}^i[v]$. Now suppose, by inductive hypothesis, that for any $0 \le j \le i-1$ the statement is true. Consider a value $v \in \mathbb{N} \cap [v_0 - n, v_0 + n]$. If $|v - (v_0 + \mathbb{E}[X^U_i] - \min\{v_0 + L^E_i, 0\})| \ge 16\sqrt{n \log n}$, then any R with $R_i = v$ is not good until time i and $\hat{\mathsf{P}}^i[v] = 0$. Otherwise if $v > 0$, we have,

$$\begin{aligned} \hat{\mathsf{P}}^i[v] &= p_i \hat{\mathsf{P}}^{i-1}[v-1] + (1-p_i)\hat{\mathsf{P}}^{i-1}[v+1] \\ &= p_i \sum_{R \in G_{i-1,v-1}} \Pr(R) \;+\; (1-p_i) \sum_{R \in G_{i-1,v+1}} \Pr(R) \end{aligned}$$

Let E_i be the event that the realization of the counter is good until time i, then $\sum_{R \in G_{i,v}} \Pr(R) = \Pr(X_{\le i} = v \wedge E_i)$. So, for $v > 0$,

$$\begin{aligned} \hat{\mathsf{P}}^i[v] &= p_i \Pr(X_{\le i-1} = v-1 \wedge E_{i-1}) \\ &\quad + (1-p_i)\Pr(X_{\le i-1} = v+1 \wedge E_{i-1}) \\ &= \Pr(X_{\le i} = v \wedge E_{i-1}) \\ &= \Pr(X_{\le i} = v \wedge E_i) \\ &= \sum_{R \in G_{i,v}} \Pr(R) \end{aligned}$$

The third equation holds because $|v - (v_0 + \mathbb{E}[X^U_{\le i}] - \min\{v_0 + L^E_i, 0\})| < 16\sqrt{n \log n}$. The case when $v = 0$ is similar. □

Now we are ready to prove Theorem 4.3.

Proof of Theorem 4.3. We implement the above algorithm by allocating $10 \log n$ bits of space for each probability value. The input probabilities are approximated to $10 \log n$ bit of precision. Since the algorithm only maintains $O(\sqrt{n \log n})$ values, the total number of bits used is $O(\sqrt{n \log(n)^3})$.

Every single calculation step involved in computing an entry $\hat{\mathsf{P}}^i[v]$ introduces an $O(\frac{1}{n^{10}})$ additive error. Since the total number of calculations performed by the algorithm is $O(n\sqrt{n \log n})$, the cumulative error caused due to the bounded precision calculations is $o(\frac{1}{n^3})$. It thus follows from Lemma 4.6 that the total variation distance between the output and the true distribution is $O(\frac{1}{n^2})$. □

4.3 Counter Independent Automata

In this section, we generalize the results of Section 4.2 to handle automata with multiple states and we also let Σ to be an arbitrary finite alphabet. As before the counter X has a barrier at 0.

We associate a directed graph G with the automaton such that the vertices of G are the states of the automaton. There is a directed edge (q, q') in G if the state q can reach the state q' within one step. We first show that if the input automaton satisfies three assumptions below, we can still compute the probability distribution of the final counter value w.r.t. $\mathcal{D}_\alpha$ given a probabilistic string $\alpha \in \mathcal{D}(\Sigma)^*_t$. Then we show that the first two assumptions can be eliminated (shown in Section 4.6), leaving only a mild assumption on probability values seen in the stream, namely, the probability values are bounded away from 0 and 1.

Assumption 1. *G is strongly connected, that is, every pair of vertices in G is connected by a path.*

Assumption 2. *G is aperiodic, that is, for every vertex s in G, GCD of the set $\{\ell \mid$ there is a walk of length ℓ from s to itself$\}$ is 1.*

Assumption 3. *There is a constant $\eta > 0$ such that for any $1 \le i \le n$ and $\sigma \in \Sigma$, $\eta < \alpha_i(\sigma) < 1 - \eta$.*

Define X_i as the value of the update $u(q,\sigma)$ at time i, define $X^U_{\le i}$, L_i, L^E_i the same as in the previous section. Let $Y_i = (X_i+1)/2$ and $Y^U_{\le i} = \sum_{j=1}^{i} Y_i$ be the number of increments seen after processing i items. We denote by S_i the random state after processing i items. For any random variable Z, let $\mathcal{D}(Z)$ be the probability distribution of Z.

Note that if the counter does not have a barrier, then the automaton is an Additive CRA; we can maintain $\mathbb{E}[X^U_{\le i}]$ and L^E_i with $\tilde{O}(1)$ space using the algorithm presented in Section 4.1, but using bounded precision arithmetic instead of buckets for approximation. However, in this setting, X_i's are not independent. In order to use the same idea as in Section 4.2, we need to prove that the distribution of $X^U_{\le i}$ is still concentrated over $\tilde{O}(\sqrt{n})$ values.

Lemma 4.7. *For any K, with probability at least $1-\frac{1}{n^3}$, $|X^U_{\le K} - \mathbb{E}[X^U_{\le K}]| \le 16\sqrt{nC}\cdot\log n$, where C is a constant that only depends on $|Q|$ and η.*

The proof relies on the following Lemma 4.8 which shows that if two positions i and j are far enough, then S_i and S_j are almost independent. We first state the lemma and prove it later in Section 4.4.

Lemma 4.8. *There exists a constant $C > 0$ which depends only $|Q|$ and η, such that for any positions i and j with $j - i \ge C\log n$, $\mathrm{TVD}(\mathcal{D}(S_j \mid S_i = q_i), \mathcal{D}(S_j \mid S_i = q_i')) = o(\frac{1}{n^3})$ for all pairs of states q_i and q_i' in Q.*

We will also use the following concentration bound which is similar to the Chernoff bounds, but allows the random variables to have low dependencies among each other. We prove this proposition in Section 4.5.

Proposition 4.9. *Let $Z = \sum_{i=1}^{N} Z_i$ where $Z_1, Z_2, \ldots, Z_N$ are N indicator random variables with the following property: for any $i \in \{1,\ldots,N\}$ and an $i-1$ dimensional binary vector $\mathbf{V}$, $|\mathbb{E}[Z_i] - \mathbb{E}[Z_i \mid \bigwedge_{j=1}^{i-1} Z_j = V_j]| < 1/N^2$. Then for any $0 < \delta < 1$, $\Pr(|Z - \mathbb{E}[Z]| > \delta N) < e^{-\delta^2 N/4}$.*

We are now ready to prove Lemma 4.7. We start by briefly reviewing the idea of the proof. By Lemma 4.8, we know that if Y_i and Y_j are "far away", then they are almost independent. We can then partition Y_i's into groups so that the random variables in each group are almost independent. We then use Proposition 4.9 to prove that the sum of the variables in each group is concentrated, which implies that $X^U_{\le i}$ is also concentrated.

Proof of Lemma 4.7. For sake of clarity, let us assume $K = n$. Let $i, j \in \{1,\ldots,n\}$ be indices with $j - i > C\log n$ where C is the same constant as in Lemma 4.8. By Lemma 4.8, for any pair of states q and q', $\mathrm{TVD}(\mathcal{D}(S_{j-1} \mid S_i = q), \mathcal{D}(S_{j-1} \mid S_i = q')) = o(\frac{1}{n^3})$, which implies that for any pair of states q and q', $\sum_{q'\in Q} |\Pr(S_{j-1} = q') - \Pr(S_{j-1} = q' \mid S_i = q)| = o(\frac{1}{n^3})$. Hence for any $y \in \{0,1\}$ and $q \in Q$,

$$
\begin{aligned}
&|\Pr(Y_j = y) - \Pr(Y_j = y \mid S_i = q)| \\
&= \Big|\sum_{q'\in Q} \Pr(Y_j = y \mid S_{j-1} = q')\cdot\Pr(S_{j-1} = q') - \Pr(Y_j = y \mid S_i = q)\Big| \\
&= \Big|\sum_{q'\in Q} \Pr(Y_j = y \mid S_{j-1} = q')\big(\Pr(S_{j-1} = q') - \Pr(S_{j-1} = q' \mid S_i = q)\big)\Big| \\
&< \frac{1}{n^3}
\end{aligned}
$$

where we have used independence of Y_j and S_i given S_{j-1}. Since for any $k \le i$, Y_j and Y_k are independent given S_i, we have, for any $y, y' \in \{0,1\}$ and $q \in Q$, $\Pr(Y_j = y \mid Y_i = y', S_i = q) = \Pr(Y_j = y \mid S_i = q)$. Now let $I \subseteq \{1,\ldots,i\}$ be a subset of indices before i. Given any $|I|$-dimensional binary vector $\mathbf{V} = (V_k)_{k\in I}$ and $y \in \{0,1\}$, we have,

$$
\begin{aligned}
&\Pr(Y_j = y \mid \bigwedge_{k\in I} Y_k = V_k) \\
&= \sum_{q\in Q} \Pr(Y_j = y \mid \bigwedge_{k\in I} Y_k = V_k, S_i = q)\cdot\Pr(S_i = q \mid \bigwedge_{k\in I} Y_k = V_k) \\
&= \sum_{q\in Q} \Pr(Y_j = y \mid S_i = q)\cdot\Pr(S_i = q \mid \bigwedge_{k\in I} Y_k = V_k) \\
&< \sum_{q\in Q} (\Pr(Y_j = y) + \frac{1}{n^3})\cdot\Pr(S_i = q \mid \bigwedge_{k\in I} Y_k = V_k) \\
&= (\Pr(Y_j = y) + \frac{1}{n^3})\cdot\sum_{q\in Q}\Pr(S_i = q \mid \bigwedge_{k\in I} Y_k = V_k) \\
&= \Pr(Y_j = y) + \frac{1}{n^3}
\end{aligned}
$$

Similarly, we have $\Pr(Y_j = y \mid \bigwedge_{k\in I} Y_k = V_k) > \Pr(Y_j = y) - \frac{1}{n^3}$, which give us, for any $i, j \in \{1,\ldots,n\}$ with $j - i > C\log n$, $y \in \{0,1\}$ and $\mathbf{V} \in \{0,1\}^I$ for some $I \subseteq \{1,\ldots,i\}$,

$$
|\Pr(Y_j = y) - \Pr(Y_j = y \mid \bigwedge_{k\in I} Y_k = V_k)| < \frac{1}{n^3} \tag{3}
$$

Let $M = \lfloor 2C\log n \rfloor$. For any $1 \le k \le M$, define $N_k = \lfloor \frac{n+(M-k)}{M} \rfloor$. Divide Y_i's into M groups as follows: For any $1 \le k \le M$ and $0 \le \ell < N_k$, let $Y^k_\ell = Y_{\ell M+k}$. Let $Y^k = \sum_{\ell=0}^{N_k-1} Y^k_\ell$.

By Inequality (3), for any $\ell > 0$, k and $V \in \{0,1\}^{\ell-1}$ have $|\Pr(Y^k_\ell = 1) - \Pr(Y^k_\ell = 1 \mid \bigwedge_{j=1}^{\ell-1} Y^k_j = V_j)| < \frac{1}{n^3} < \frac{1}{N_k^3}$. By Proposition 4.9, for any k, with probability at least $1 - \frac{1}{n^4}$, $|Y^k - \mathbb{E}[Y^k]| \le 4\sqrt{N_k \log n}$.

Take union bound over k, with probability at least $1 - \frac{1}{n^3}$, we have,

$$
\begin{aligned}
|Y^U_{\le n} - \mathbb{E}[Y^U_{\le n}]| &\le \sum_{k=1}^{M} |Y^k - \mathbb{E}[Y^k]| \\
&\le 4\sum_{k=1}^{M}\sqrt{N_k\log n} \\
&\le 8\sqrt{C\log n}\cdot\sqrt{n\log n}
\end{aligned}
$$

$$= 8\sqrt{nC} \cdot \log n$$

where the last inequality follows from Cauchy-Schwarz. Realizing that $X^U_{\leq n} = 2Y^U_{\leq n} - 1$, we get the required bound. □

By replacing the use of the Chernoff bounds with Lemma 4.7 in the proof of Lemma 4.5, we get that with probability at least $1 - \frac{1}{n^2}$, for any $1 \leq i \leq n$, $X_{\leq i}$ lies within an $O(\sqrt{nC}\log n)$-sized band around $v_0 + \mathbb{E}[X^U_{\leq i}] - \min\{v_0 + L^E_i, 0\}$. Therefore we can maintain an approximate distribution of $(S_i, X_{\leq i})$ using $O(|Q|\sqrt{nC}\log n)$ values where each value takes $O(\log n)$ bits of space. $\mathbb{E}[X^U_{\leq i}]$ and L^E_i can be maintained using $O(|Q|\log n)$ space by maintaining $\mathbb{E}[X^U_{\leq i} \mid S_i = q]$ for every $q \in Q$. Therefore, using similar arguments as in the proof of Theorem 4.3 we obtain the following result.

Lemma 4.10. *For any CRA $\mathcal{A}$ with a single counter (with barrier at 0) having updates from the set {+1,-1} with the underlying graph being strongly connected and aperiodic, there is a single pass streaming algorithm that, given a probabilistic string $\alpha \in \mathcal{D}(\Sigma)^*_t$ satisfying Assumption 3, computes a probability distribution over the counter values at the end of the stream whose total variation distance from the true distribution of the counter values is $O(\frac{1}{n^2})$ and uses $O(C'\sqrt{n} \cdot \log(n)^2 + t)$ space, where C' depends only on $|\mathcal{A}|$ and η.*

4.4 Proof of Lemma 4.8

We start with a brief overview of the proof. The main ingredient in the proof is to utilize the coupling technique which is often used in proving results about the mixing time of Markov Chains. In this technique, one shows that two independent runs that may start at different states end up meeting within a certain bounded period of time. The executions can then be coupled from here on. We show that Assumptions 1 through 3 guarantee such a coupling. In particular, Assumptions 1 and 2 guarantee that for any two runs it is possible to reach the same state in a short period of time, and Assumption 3 guarantees that this event happens with high probability.

We start with a simple proposition about strongly connected aperiodic graphs.

Proposition 4.11. *If a directed graph $G = (V, E)$ is strongly connected and aperiodic, then for any vertices v_a, v_b, v_c, there is an integer L such that both v_a and v_b could reach v_c in L steps.*

Proof. Since the graph G is aperiodic, the GCD of the lengths of walks from v_c to itself is 1. Let $Wlk_1, Wlk_2, \ldots, Wlk_K$ be a set of walks from v_c to itself such that the GCD of their lengths is 1. By Bézout's identity, there is an integer vector $(\ell_1, \ell_2, \ldots \ell_K)$ such that $\sum_{i=1}^K \ell_i |Wlk_i| = 1$. Let d_a and d_b be the distance from v_a and v_b to v_c respectively. If $d_a = d_b$, then the statement immediately follows. Otherwise, suppose $d_a < d_b$. Consider the path starting from d_a as follows: we first go to v_c with the path of length d_a, then for any i such that $\ell_i > 0$, we go through Wlk_i and back to v_c, $\ell_i(d_b - d_a)$ times. Similarly, consider the path from v_b as follows: we first go to v_c along the path of length d_b, then for any i such that $\ell_i < 0$, we traverse Wlk_i and back to v_c, $-\ell_i(d_b - d_a)$ times. The difference between the lengths of these two paths is $d_a - d_b + (d_b - d_a)\sum_{i=1}^K \ell_i |Wlk_i| = 0$. □

We use a coupling argument to prove Lemma 4.8. Let $i \in \{1, \ldots, n\}$ and $q_i, q'_i \in Q$ be any two states. Consider the following probabilistic process: Simulate two simultaneous runs over the automaton, starting at q_i and q'_i respectively. Process symbols from position i onwards, updating the states according to the transitions of the automaton. At any time $k \geq i$, if the two runs are at different states, then the next symbols showing up in the two runs are independently sampled from α_k. However, if they are at the same state, the next symbols showing up in the two runs are jointly sampled from α_k, and hence are identical (ensuring that from now on, the two runs remain in the same state). For any pair of states q, q' in Q and time $k \geq i$, we define $P^k_{q,q'}$ to be the probability of the two runs being in states q and q', respectively, at time k. Formally, $P^k_{q,q'}$ is defined inductively as follows:

- $P^i_{q,q'} = \begin{cases} 1 & \text{if } q = q_i \text{ and } q' = q'_i \\ 0 & \text{otherwise} \end{cases}$
- If $q \neq q'$ and $k > i$,

$$P^k_{q,q'} = \sum_{s \neq s'} P^{k-1}_{s,s'} \cdot \Pr(S_k = q \mid S_{k-1} = s) \cdot \Pr(S_k = q' \mid S_{k-1} = s')$$

- If $q = q'$ and $k > i$,

$$P^k_{q,q'} = \sum_{s \neq s'} P^{k-1}_{s,s'} \cdot \Pr(S_k = q \mid S_{k-1} = s) \cdot \Pr(S_k = q' \mid S_{k-1} = s') + \sum_s P^{k-1}_{s,s} \cdot \Pr(S_k = q \mid S_{k-1} = s)$$

With this definition, we make the following two observations.

Observation 1. *For any time $k \geq i$ and state $q \in Q$, $\Pr(S_k = q \mid S_i = q_i) = \sum_{q'} P^k_{q,q'}$, similarly, for any $q' \in Q$, $\Pr(S_k = q' \mid S_i = q'_i) = \sum_q P^k_{q,q'}$.*

Observation 2. *The probability that two independent runs starting from q_i and q'_i at time i meet each other in the same state before time k is $\sum_q P^k_{q,q}$.*

We now complete the remaining details of the proof.

Proof of Lemma 4.8. By Observation 1,

$$\begin{aligned} &\mathrm{TVD}(\mathcal{D}(S_j \mid S_i = q_i), \mathcal{D}(S_j \mid S_i = q'_i)) \\ &= \frac{1}{2}\sum_q |\Pr(S_j = q \mid S_i = q_i) - \Pr(S_j = q \mid S_i = q'_i)| \\ &= \frac{1}{2}\sum_q \left| \sum_{q'} P^j_{q,q'} - \sum_{q'} P^j_{q',q} \right| \end{aligned}$$

$$\leq \frac{1}{2}\sum_{q}\Big(\sum_{q'\neq q} P^j_{q,q'} + \sum_{q'\neq q} P^j_{q',q}\Big)$$
$$= 1 - \sum_{q} P^j_{q,q}$$

Let $C = 4|Q|^2(\frac{1}{\eta})^{2L(|Q|)}$ where $L(|Q|)$ is the upper bound of L in Proposition 4.11 when the graph G has $|Q|$ vertices. By Observation 2, it is sufficient to prove that two independent runs starting from q_i and q'_i at time i meet each other before time j has probability $1 - o(\frac{1}{n^3})$ if $j - i \geq C\log n$. Fix an arbitrary state q, by Proposition 4.11, there exists two paths to q from any two states, with the same length which is at most $L(|Q|)$. So during any time period of length $L(|Q|)$, two independent runs meet with each other with probability at least $\eta^{2L(|Q|)}$ by Assumption 3. Since $L(|Q|) \leq |Q|^2$, if $j - i \geq C\log n$, there are at least $4(\frac{1}{\eta})^{2L(|Q|)}\log(n)$ such non-overlapping time periods. With probability $1 - o(\frac{1}{n^3})$, two independent runs meet with each other during at least one of these time periods. □

4.5 Proof of Proposition 4.9

The main idea is to construct a Doob martingale and use Azuma's inequality to obtain the concentration bound.

For any integer $0 \leq i \leq N$, let $\mathcal{F}_i$ be the σ-algebra of set $\{Z_1, Z_2, \ldots, Z_i\}$. Then for any $0 \leq i < j \leq N$, $\mathcal{F}_i \subset \mathcal{F}_j$, which implies that $\{\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_N\}$ is a filtration. Let $\bar{Z}_i = \mathbb{E}[Z \mid \mathcal{F}_i]$ for any $0 \leq i \leq N$, $\{\bar{Z}_0, \bar{Z}_1, \ldots, \bar{Z}_N\}$ is a Doob martingale.

Then, for any $1 \leq i \leq N$, we have

$$|\bar{Z}_i - \bar{Z}_{i-1}| = |\mathbb{E}[Z \mid \mathcal{F}_i] - \mathbb{E}[Z \mid \mathcal{F}_{i-1}]|$$
$$= \Big|\sum_{j=1}^{N}(\mathbb{E}[Z_j \mid \mathcal{F}_i] - \mathbb{E}[Z_j \mid \mathcal{F}_{i-1}])\Big|$$
$$\leq \sum_{j=1}^{N}|\mathbb{E}[Z_j \mid \mathcal{F}_i] - \mathbb{E}[Z_j \mid \mathcal{F}_{i-1}]|$$

When $j < i$,

$$\mathbb{E}[Z_j \mid \mathcal{F}_i] = \mathbb{E}[Z_j \mid \mathcal{F}_{i-1}]$$

When $j = i$,

$$|\mathbb{E}[Z_j \mid \mathcal{F}_i] - \mathbb{E}[Z_j \mid \mathcal{F}_{i-1}]| \leq 1$$

since Z_j is an indicator variable.

When $j > i$,

$$|\mathbb{E}[Z_j \mid \mathcal{F}_i] - \mathbb{E}[Z_j \mid \mathcal{F}_{i-1}]|$$
$$\leq |\mathbb{E}[Z_j \mid \mathcal{F}_i] - \mathbb{E}[Z_j]| + |\mathbb{E}[Z_j \mid \mathcal{F}_{i-1}] - \mathbb{E}[Z_j]|$$
$$\leq \frac{2}{N^2}$$

where the last inequality holds because for any $j > i$,

$$|\mathbb{E}[Z_j \mid \mathcal{F}_i] - \mathbb{E}[Z_j]| = \mathop{\mathbb{E}}_{Z_{i+1},\ldots,Z_{j-1}}\big[|\mathbb{E}[Z_j \mid \mathcal{F}_{j-1}] - \mathbb{E}[Z_j]|\big]$$
$$< \frac{1}{N^2}.$$

So,

$$|\bar{Z}_i - \bar{Z}_{i-1}| \leq 1 + \sum_{j=i+1}^{N}\frac{2}{N^2} \leq 1 + \frac{2}{N}$$

By Azuma's inequality, for any $\gamma > 0$,

$$\Pr(|\bar{Z}_N - \bar{Z}_0| \geq \gamma) \leq \exp\left(\frac{-\gamma^2}{2\sum_{i=1}^{N}(1+\frac{2}{N})^2}\right) < e^{\frac{-\gamma^2}{4N}}$$

Note that $\bar{Z}_N = Z$ and $\bar{Z}_0 = \mathbb{E}[Z]$. Therefore, letting $\gamma = \delta N$, we have

$$\Pr(|Z - \mathbb{E}[Z]| \geq \delta N) < e^{\frac{-\delta^2 N}{4}}$$

□

4.6 Eliminating the first two Assumptions

In this section, we describe how to eliminate the first two assumptions on the structure of the automaton. Note that we only use the first two assumptions in the proof of Lemma 4.8, so we only need to show how to modify the proof of this lemma alone to make the argument still work without these two assumptions.

Eliminating the Aperiodicity Assumption

We say that the period associated with a vertex s_a in G is γ if GCD of the set $\{\ell \mid$ there is a walk of length ℓ from s_a to itself$\}$ is γ. Since G is strongly connected, every vertex has the same period, say $K > 1$, which is the period of the graph G. We first form K sets of states $Q_0, Q_1, \ldots Q_{K-1}$ with $\bigcup_\ell Q_\ell = Q$ such that $q_0 \in Q_0$ and any state in Q_ℓ can only reach a state in $Q_{(\ell+1) \mod K}$ in one step. We now construct graphs $G_0, G_1, \ldots, G_{K-1}$ as follows: the vertex set of G_ℓ is Q_ℓ and for any pair of states q and q' in G_ℓ, there is an edge (q, q') in G_ℓ if and only if q can reach q' in exactly K steps in G. Since K is the period of G, each G_ℓ is necessarily an aperiodic graph. If q has an edge to q' in G_ℓ, then by Assumption 3, if the automaton is in state q at some time, then after K steps, the automaton is in state q' with probability at least η^K. Using a similar argument as in the proof of Lemma 4.8, we obtain the following lemma.

Lemma 4.12. *Suppose the input satisfies Assumptions 1 and 3, then there is a constant $C > 0$ which depends only on $|Q|$ and η such that, for any times i and j with $j - i \geq CK\log n$, $\mathrm{TVD}(\mathcal{D}(S_j|S_i = q_i), \mathcal{D}(S_j|S_i = q'_i)) = o(\frac{1}{n^3})$ for any pair of states q_i and q'_i in $Q_{(i \mod K)}$.*

Then using the same argument as in Lemma 4.10, we have an algorithm to solve the problem if the input satisfies Assumptions 1 and 3.

Lemma 4.13. *For any CRA $\mathcal{A}$ with a single counter (with barrier at 0) having updates from the set {+1,-1} with the underlying graph being strongly connected with period K, there is a single pass streaming algorithm that, given a probabilistic string $\alpha \in \mathcal{D}(\Sigma)^*_t$ satisfying Assumption 3, computes a probability distribution over the counter values at the end of the*

stream whose total variation distance from the true distribution of the counter values is $O(\frac{1}{n^2})$ and uses $O(C'\sqrt{Kn}\cdot\log(n)^2+t)$ space where C' depends only on $|\mathcal{A}|$ and η.

Eliminating the Strongly Connected Assumption

Suppose the graph G is not strongly connected, let Q' be the union of all strongly connected components of G that have out-degree 0 in the SCC graph of G. By Assumption 3, for each time period of length $|Q|$, with probability $\eta^{|Q|}$, the automata will reach a state in Q'. So after $3(\frac{1}{\eta})^{|Q|}\log n$ steps, the automata is at a state in Q' with probability at least $1-o(\frac{1}{n^2})$. Once the automata reaches a state in an SCC of out-degree 0, it cannot leave such an SCC, so we only need to focus on this SCC. We formalize this idea in what follows.

In the first $3(\frac{1}{\eta})^{|Q|}\log n$ steps, we maintain a table $T : Q\times[v_0-3(\frac{1}{\eta})^{|Q|}\log n, v_0+3(\frac{1}{\eta})^{|Q|}\log n]\to[0,1]$ where the entry $T(q,k)$ stores the probability that the automaton is at state q and the counter has value k. At time $3(\frac{1}{\eta})^{|Q|}\log n$, for any state q in Q', we solve the problem when initial state is q and $v_0=0$. Instead of maintaining the distribution of the counter itself, we maintain the distribution of L_i and $X^U_{\le i}$. By Lemma 4.13, we can compute these two distributions with $O(\sqrt{n}\cdot\log^2 n)$ space. To distinguish between different q's, denote these two values as L_i^q and $X^{U,q}_{\le i}$. Then for any value $k\in[v_0-3(\frac{1}{\eta})^{|Q|}\log n, v_0+3(\frac{1}{\eta})^{|Q|}\log n]$, let $X^{q,k}_{\le n}$ be the value of the counter if we start at q with initial value k. By Lemma 4.4, we can compute the distribution of $X^{q,k}_{\le n}$ with the distribution of L_i^q and $X^{U,q}_{\le i}$. Finally, output $\mathcal{D}(X_{\le n}) = \sum_{q,k} T(q,k)\cdot\mathcal{D}(X^{q,k}_{\le n})$.

For each q, storing $X^{U,q}_{\le i}$ and L_i^q requires $O(\sqrt{n}\log^2 n)$ bits of space. Each $\mathcal{D}(X^{q,k}_{\le n})$ also requires $O(\sqrt{n}\log^2 n)$ bits of space and there are $|Q|\cdot 6(\frac{1}{\eta})^{|Q|}\log n$ pairs of q and k, so storing all $\mathcal{D}(X^{q,k}_{\le n})$ requires $O(\sqrt{n}\log^3 n)$ bits of space. T has $|Q|\cdot 6|Q|(\frac{1}{\eta})^{|Q|}\log n = O(\log n)$ entries. So storing T requires $O(\log^2 n)$ bits of space.

Theorem 4.14. *For any CRA $\mathcal{A}$ with a single counter (with barrier at 0) having updates from the set {+1,-1}, there is a single pass streaming algorithm that, given a probabilistic string $\alpha\in\mathcal{D}(\Sigma)^*_t$ satisfying Assumption 3, computes a probability distribution over the counter values at the end of the stream whose total variation distance from the true distribution of the counter values is $O(\frac{1}{n^2})$ and uses $O(C'\sqrt{n}\cdot\log(n)^3+t)$ space, where C' depends only on $|\mathcal{A}|$ and η.*

5 Conclusions

We have studied the space complexity of single-pass streaming computation for different classes of queries over probabilistic strings. In particular, we designed a poly-logarithmic space algorithm for estimating the probability of a regular pattern appearing in a probabilistic event stream, to within an arbitrary specified precision. We also developed an $\tilde{O}(\sqrt{n})$ space streaming algorithm for computing the distribution profile for queries that involve conditional updates to a counter variable. Our result is based on a technical insight that may be of independent interest, namely, at any point in the stream, the range of counter values can be narrowed down with high probability to a band of $\tilde{O}(\sqrt{n})$ values.

Future work and Open problems

Recent work on monitoring systems for quantitative queries has focused on design, implementation, and empirical evaluation of query languages over streaming data where theoretical foundations allow the compiler to provide guarantees on resource usage [9, 14]. A natural next step is to extend these query languages to allow computation over probabilistic data streams based on the theoretical insights of this paper.

We assumed throughout the paper that a bound on the length n of the input stream is known in advance. If such a bound is not known a priori, we can use standard doubling idea to extend our results to handle streams with unknown length.

Another interesting direction for future work is to prove non-trivial space lower bounds for single-pass streaming algorithms for computational problems studied in this paper. For example, for the problem considered in section 3.1, what is the space complexity for an *exact* single-pass streaming algorithm to determine whether or not the product of the input numbers is below a specified threshold value. If we allow the input stream to contain arbitrary rational numbers, instead of rational numbers in the interval $[0,1]$ as is the case in our context, then the $\Omega(n)$ communication bound for Equality (see, [8], for instance) can be adapted to show an $\Omega(n)$ lower bound on the space required for a deterministic single-pass streaming algorithm (see Appendix A.2). However, a lower bound in our setting remains an open problem. Finally, our results on handling conditional counter updates (for instance, Theorem 4.3) raise the question of whether the upper bound $\tilde{O}(\sqrt{n})$ on space usage is optimal. We conjecture that any single-pass streaming algorithm that computes the probability distribution over counter values to within an inverse polynomial precision (i.e. TVD from the actual distribution is inverse polynomially small), requires $\Omega(\sqrt{n})$ space.

Acknowledgements. We thank the anonymous reviewers for their insightful comments. This research was partially supported by NSF awards CCF-1763514, CCF-1723567, CCF-1617851, CCF-1763514 and CCF-1934876.

References

[1] Noga Alon, Yossi Matias, and Mario Szegedy. 1999. The Space Complexity of Approximating the Frequency Moments. *J. Comput. Syst. Sci.* 58, 1 (1999), 137–147.

[2] Rajeev Alur, Loris D'Antoni, Jyotirmoy Deshmukh, Mukund Raghothaman, and Yifei Yuan. 2013. Regular Functions and Cost Register Automata. In *Proceedings of the 28th Annual ACM/IEEE Symposium on Logic in Computer Science*. 13–22.

[3] Kevin Borders, Jonathan Springer, and Matthew Burnside. 2012. Chimera: A Declarative Language for Streaming Network Traffic Analysis. In *Proceedings of the 21st USENIX Conference on Security Symposium*. 19–19.

[4] Graham Cormode and Minos Garofalakis. 2007. Sketching Probabilistic Data Streams. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*. 281–292.

[5] T. S. Jayram, Satyen Kale, and Erik Vee. 2007. Efficient Aggregation Algorithms for Probabilistic Data. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. 346–355.

[6] T. S. Jayram, Andrew McGregor, S. Muthukrishnan, and Erik Vee. 2007. Estimating Statistical Aggregates on Probabilistic Data Streams. In *Proceedings of the Twenty-sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. 243–252.

[7] Cheqing Jin, Ke Yi, Lei Chen, Jeffrey Xu Yu, and Xuemin Lin. 2010. Sliding-window Top-k Queries on Uncertain Streams. *The VLDB Journal* 19, 3 (2010), 411–435.

[8] Eyal Kushilevitz and Noam Nisan. 2006. *Communication Complexity*. Cambridge University Press.

[9] Konstantinos Mamouras, Mukund Raghothaman, Rajeev Alur, Zachary G. Ives, and Sanjeev Khanna. 2017. StreamQRE: Modular Specification and Efficient Evaluation of Quantitative Queries over Streaming Data. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 693–708.

[10] Michael Mitzenmacher and Eli Upfal. 2005. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press.

[11] S. Muthukrishnan. 2005. Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science* 1, 2 (2005).

[12] Vern Paxson. 1999. Bro: A System for Detecting Network Intruders in Real-time. *Computer Networks* 31, 23-24 (1999), 2435–2463.

[13] Christopher Ré, Julie Letchner, Magdalena Balazinksa, and Dan Suciu. 2008. Event Queries on Correlated Probabilistic Streams. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. 715–728.

[14] Yifei Yuan, Dong Lin, Ankit Mishra, Sajal Marwaha, Rajeev Alur, and Boon Thau Loo. 2017. Quantitative Network Monitoring with NetQRE. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM*. 99–112.

A Appendix

A.1 Proof of Lemma 4.1

Lemma. *For all $i > 0$,*

$$\mathbf{E}^i(q) = \sum_{\substack{q',\sigma \\ \delta(q',\sigma)=q}} \alpha_i(\sigma)\Big[\mathbf{M}(q',\sigma)\mathbf{E}^{i-1}(q') + \mathsf{P}^{i-1}(q')\mathbf{u}(q',\sigma)\Big]$$

Proof. From Equation 1,

$$\begin{aligned}
\mathbf{E}^i(q) &= \sum_{\substack{|w'|=i \\ \delta(q_0,w')=q}} \mathcal{D}_\alpha(w')\mathbf{v}(w') \\
&= \sum_{|w''|=i-1} \sum_{\substack{\sigma\in\Sigma \\ \delta(q_0,w''\sigma)=q}} \mathcal{D}_\alpha(w''\sigma)\mathbf{v}(w''\sigma) \\
&= \sum_{q'\in Q} \sum_{\substack{|w''|=i-1 \\ \delta(q_0,w'')=q'}} \sum_{\substack{\sigma\in\Sigma \\ \delta(q',\sigma)=q}} \mathcal{D}_\alpha(w'')\alpha_i(\sigma)\Big[\mathbf{M}(q',\sigma)\mathbf{v}(w'') + \mathbf{u}(q',\sigma)\Big] \\
&= \sum_{q'\in Q} \sum_{\substack{\sigma\in\Sigma \\ \delta(q',\sigma)=q}} \alpha_i(\sigma) \sum_{\substack{|w''|=i-1 \\ \delta(q_0,w'')=q'}} \Big[\mathbf{M}(q',\sigma)\Big(\mathcal{D}_\alpha(w'')\mathbf{v}(w'')\Big) + \mathcal{D}_\alpha(w'')\mathbf{u}(q',\sigma)\Big] \\
&= \sum_{\substack{q',\sigma \\ \delta(q',\sigma)=q}} \alpha_i(\sigma)\Big[\mathbf{M}(q',\sigma)\Big(\sum_{\substack{|w''|=i-1 \\ \delta(q_0,w'')=q'}} \mathcal{D}_\alpha(w'')\mathbf{v}(w'')\Big) + \Big(\sum_{\substack{|w''|=i-1 \\ \delta(q_0,w'')=q'}} \mathcal{D}_\alpha(w'')\Big)\mathbf{u}(q',\sigma)\Big] \\
&= \sum_{\substack{q',\sigma \\ \delta(q',\sigma)=q}} \alpha_i(\sigma)\Big[\mathbf{M}(q',\sigma)\mathbf{E}^{i-1}(q') + \mathsf{P}^{i-1}(q')\mathbf{u}(q',\sigma)\Big]
\end{aligned}$$

□

A.2 Space Lower Bound for Exactly Computing Product

Theorem A.1. *On input a stream of rational numbers $q_1, ..., q_n$ such that each q_i can be represented in binary using $O(\log(n))$ bits and a threshold value τ, any deterministic single-pass streaming algorithm to decide if $\Pi_{i=1}^n q_i < \tau$ requires $\Omega(n)$ space.*

Proof. Let $p_1, ..., p_n$ denote the first n prime numbers. Let t_n be the number of bits required to represent all the values in $\{p_1, ..., p_n\} \cup \{p_1^{-1}, ..., p_n^{-1}\}$ and let $\tau = 1$. From the prime number thoerem we know that $t_n = O(\log(n))$. Consider streams of length $2n$. Given any two sequences for the first half with values from $\{p_1^{-1}, ..., p_n^{-1}\}$ such that the product of the two sequences are different, there is a sequence for the second half with values from $\{p_1, ..., p_n\}$ such that the resulting stream has product 1 in one case and less that 1 in the other case. Hence, any streaming algorithm that correctly answers the threshold question needs $\Omega(n)$ space because, for large enough n, there are $\binom{2n-1}{n} \geq 2^{n-1}$ different possible products for the first half and we need atleast $\log(2^{n-1})$ space to distinguish them. □