Verification of the CVM algorithm with a Functional Probabilistic Invariant

Anonymous author(s)
Anonymous affiliation(s)

Abstract

Estimating the number of distinct elements in a data stream is a classic problem with numerous applications in computer science. We formalize a recent, remarkably simple, randomized algorithm for this problem due to Chakraborty, Vinodchandran, and Meel (called the CVM algorithm). Their algorithm deviated considerably from the state of the art, due to its avoidance of intricate derandomization techniques, while still maintaining a close-to-optimal logarithmic space complexity.

Central to our formalization is a new proof technique based on *functional probabilistic invariants*, which allows us to derive concentration bounds using the Cramér–Chernoff method without relying on independence. This simplifies the formal analysis considerably compared to the original proof by Chakraborty et al. Moreover, our technique opens up the possible algorithm design space; we demonstrate this by introducing and verifying a new variant of the CVM algorithm that is both total and unbiased—neither of which are properties of the original algorithm. In this paper, we introduce the proof technique, describe its use in mechanizing both versions of the CVM algorithm in Isabelle/HOL, and present a supporting formalized library on negatively associated random variables used to verify the latter variant.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Higher order logic; Mathematics of computing → Probabilistic algorithms; Theory of computation → Pseudorandomness and derandomization

Keywords and phrases Verification, Isabelle/HOL, Randomized Algorithms, Distinct Elements

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

In 2022, Chakraborty, Vinodchandran, and Meel [9] published a marvelous streaming algorithm for the distinct elements problem which was very unexpected in the community [36]. Indeed, Knuth later wrote a note on the algorithm [29], pointing out its interesting properties and christening it the *CVM* algorithm (which we use for the rest of this paper). One striking property of the CVM algorithm is that, in contrast to every other known algorithm for the problem, it does not rely on hashing the stream elements. Instead, the algorithm could theoretically be implemented in a setting where objects in the data stream only allow for equality comparisons. Another property is its simplicity—which is why the authors called it "an algorithm for the text book"—the algorithm is displayed in its entirety in Algorithm 1.

The pen-and-paper analysis of CVM [9, 10] relies on a sequence of transformations of the algorithm. The reason for these transformations is that standard methods for analyzing randomized algorithms, such as Chernoff–Hoeffding bounds, usually make statements about independent random variables. However, for Algorithm 1, the state variables are far from being independent.[1] For example, in Line 3 the Bernoulli distribution is sampled for the value p, which itself depends on previous random operations; similarly, the subsampling step in Line 9 is only applied if the buffer χ is full, which also depends on previous random operations.

[1] There is an incorrect claim in the initial published proof of CVM [9, Claim 6] that the indicator functions for elements in χ are independent; a later version by the same authors [10] provides a correct proof. The original error serves as a side motivation for this work.

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Algorithm 1 CVM algorithm for distinct elements estimation [9].

Input: Stream elements $a_1, \ldots, a_l$, $0 < \varepsilon$, $0 < \delta < 1$.
Output: A cardinality estimate R for set $A = \{a_1, \ldots, a_l\}$ s.t. $\mathcal{P}\left(|R - |A|| > \varepsilon |A|\right) \leq \delta$

1: $\chi \leftarrow \{\}, p \leftarrow 1, n = \left\lceil \frac{12}{\varepsilon^2} \ln\left(\frac{6l}{\delta}\right) \right\rceil$
2: **for** $i \leftarrow 1$ to l **do**
3: $\quad b \xleftarrow{\$} \mathrm{Ber}(p)$ ▷ insert a_i with probability p (and remove it otherwise)
4: $\quad$ **if** b **then**
5: $\qquad \chi \leftarrow \chi \cup \{a_i\}$
6: $\quad$ **else**
7: $\qquad \chi \leftarrow \chi - \{a_i\}$
8: $\quad$ **if** $|\chi| = n$ **then** ▷ if buffer χ is full
9: $\qquad \chi \xleftarrow{\$} \mathrm{subsample}(\chi)$ ▷ discard elements of χ independently with prob. $\frac{1}{2}$
10: $\qquad p \leftarrow \frac{p}{2}$
11: $\quad$ **if** $|\chi| = n$ **then return** $\perp$ ▷ fail if χ remains full
12: **return** $\frac{|\chi|}{p}$ ▷ estimate cardinality of A

The aforementioned sequence of transformations by Chakraborty et al. results in another randomized algorithm which can be analyzed using standard methods, and from which the desired results for the original algorithm can be deduced. To our knowledge, it seems impossible to analyze Algorithm 1 more directly using known textbook methods [3, 34, 35].

In this paper, we present a new technique for analyzing randomized algorithms which yields a direct and substantially more general proof of the CVM algorithm. Our approach is very similar to how deterministic algorithms are verified using loop invariants. The key difference is that our choice of “loop invariant” for the randomized streaming algorithm is a functional probabilistic inequality, namely, we consider invariants of the form:

$$\mathrm{E}[h] \leq h(c)$$

where h is allowed to range over a class of functions, and the expectation is taken over the distribution of the state of the algorithm after consuming each stream element. By first establishing such an invariant for Algorithm 1, we can then use it (via different choices of h) to establish error bounds for the algorithm. We believe the new proof remains accessible at the undergraduate level, albeit with some exposure to mechanized theorem proving.

To show the generality of our technique, we introduce a new variant of the CVM algorithm, where the subsampling step in Line 9 of Algorithm 1 selects a random m-subset of χ instead of independently discarding each element with some probability. This variant has the benefit that it is *total* (never returns $\perp$) because the second check in Line 11 becomes obsolete. More interestingly, the resulting variant is *unbiased*, i.e., the expected result is exactly the cardinality of the elements in the stream; this is a new property that neither the original CVM algorithm nor classic algorithms for the distinct elements problem possess.

The modified subsampling step leads to additional dependence for the elements in χ which cannot be readily removed using transformations following the original proof. Instead, we verify the new variant with our probabilistic invariant-based approach, using results from the theory of negatively associated random variables [25] to establish the desired functional invariant. The concept of negative association is a generalization of independence; importantly, negatively associated variables observe closure properties and fulfill Chernoff–Hoeffding bounds similar to independent random variables. It should be stressed that the theory of negatively association is orthogonal to our new technique, but its formalization is also a contribution of this work.

In summary, our main contributions are:

- Introduction of a new technique using functional probabilistic invariants to verify randomized algorithms inductively/recursively.
- Verification of the original CVM algorithm using our new technique.
- Presentation and verification of a new variant of CVM that is total and unbiased.
- Formalization of a theory of negatively associated random variables used to analyze the new CVM variant.

We carried out the mechanizations using Isabelle/HOL [37], which comes with a large repository of foundational libraries [1] for the verification of randomized algorithms. We have also mechanized the transformation-based CVM proof by Chakraborty et al. [9, 10], which provides a rough point of comparison: verification of the CVM algorithm using our new technique required only 1003 lines, while the original proof required 2630 lines.

The rest of this paper is organized as follows. Section 2 provides background information on randomized algorithms, in particular on their semantics in Isabelle/HOL. Section 3 introduces our new technique and explains how probabilistic loop invariants can be used to establish tail bounds for the original CVM algorithm. Section 4 introduces the concept of negative association and our new total and unbiased variant of the CVM algorithm. Section 5 presents the formalization of both variants of the algorithm, and Section 6 describes our new formalized library on negatively associated random variables. Section 7 discusses some challenges faced in our alternative verification of CVM using the transformation-based proof by Chakraborty et al. The final sections present related work and a summary of our results.

The supplementary material contains:

- formalization of the CVM algorithm, both the original version (Algorithm 1) and our new version (Algorithm 3) using functional probabilistic invariants;
- formalization of a library for negative association; and
- formalization of the CVM algorithm following the proof by Chakraborty et al. [9, 10].

2 Background

2.1 Randomized Algorithms and Distinct Elements

The CVM algorithm is a *streaming* algorithm for the distinct elements problem. As shown in Algorithm 1, given a data stream $a_1, \ldots, a_l$, the goal of such algorithms is to return an accurate cardinality estimate for the set $A = \{a_1, \ldots, a_l\}$.

Importantly, CVM is a *probably-approximately correct* (PAC) algorithm where its output estimate R satisfies $\mathcal{P}\left(|R - |A|| > \varepsilon|A|\right) \leq \delta$ for parameters ε and δ, i.e., the probability that the relative error of R with respect to $|A|$ exceeds ε is at most δ. Moreover, let us assume the space needed to store each element in the stream is b bits, then the CVM algorithm requires only $\mathcal{O}(\varepsilon^{-2} b \ln(\delta^{-1} l))$ bits of mutable state, which is far less than storing each stream element deterministically.

▶ Remark 1. The optimal randomized algorithm for distinct elements requires $\mathcal{O}(\varepsilon^{-2} \ln \delta + b)$ bits, but it requires more advanced algorithmic techniques. It would not be possible to present using such elementary steps as in Algorithm 1 as it involves computations in finite fields and random walks in expander graphs [7, 28]. ⌟

2.2 Semantics of Randomized Algorithms

We briefly review how reasoning about randomized algorithms works in Isabelle/HOL using the Giry monad [19]. A thorough discussion of the concept in the context of Isabelle/HOL has been written, for example, by Eberl et al. [15] and Lochbihler [31].

The key idea is to model a randomized algorithm as a probability space representing the distribution of its results. As an example, let us consider Algorithm 2.

Algorithm 2 Example for sequential composition.

1: $p \xleftarrow{\$} \mathrm{Ber}(\frac{1}{2})$
2: $q \xleftarrow{\$} \mathrm{Ber}(\frac{1}{3} + \frac{p}{2})$
3: **return** q

In the first step, Algorithm 2 flips a fair coin, such that p is 1 with probability $\frac{1}{2}$ and 0 otherwise; the notation $\mathrm{Ber}(p)$ represents the Bernoulli distribution. In the second step, the algorithm flips a coin q which depends on p. This has the consequence that, to semantically model q, we have to consider probability space-valued functions, like: $p \mapsto \mathrm{Ber}(\frac{1}{3} + \frac{p}{2})$, which is being *bound* to the distribution of p. The resulting distribution for q is a *compound distribution* resulting from a combination of $\mathrm{Ber}(\frac{1}{3})$ (when $p = 0$) and $\mathrm{Ber}(\frac{5}{6})$ (when $p = 1$).

This example captures the main aspects of modeling randomized algorithms in the Giry monad. Indeed, randomized algorithms can be modeled using the following ingredients:

Primitive Random Operations. For example, a simple fair coin flip is represented using the Bernoulli distribution, $\mathrm{Ber}(\frac{1}{2})$.

Return Combinator. Given a singleton element x, we can construct the singleton probability space, assigning probability 1 to x and 0 to everything else. In monad notation, this is written as: $\mathbf{return}\, x$.

Bind Combinator. The bind combinator represents sequential composition of two randomized algorithms m and f, where the latter randomized algorithm consumes the output of the former; in monad notation, this is: $m \gg\!= f$. Mathematically, this is the most involved operation, because f is a function returning probability spaces, which takes inputs from the probability space m.

Let us consider an event A in the probability space $m \gg\!= f$. Its probability can be evaluated by integrating over its probabilities in f with respect to m:

$$\mathcal{P}_{m \gg= f}(A) = \int_m \mathcal{P}_{f(x)}(A)\, dx.$$

Another key property is the calculation of expectations; if h is a random variable over $m \gg\!= f$, we can compute its expectation as:

$$\mathrm{E}_{m \gg= f}[h] = \int_m \mathrm{E}_{f(x)}[h]\, dx. \tag{1}$$

Equation (1) is crucially used to establish the invariants we introduce next in Section 3.

3 Functional Probabilistic Invariants

In this section, we will derive our new technique using Algorithm 1 as an example. Let us start by briefly reviewing the algorithm—its state is a buffer χ (initially empty) and a fraction $p > 0$ (initially set to 1). The buffer tracks a subset of the elements of the stream encountered so far, with maximal size n chosen according to the desired accuracy parameters ε, δ, and the stream size l. The algorithm iterates over the stream elements, adding each one to the buffer with probability p or conversely—if the current stream element is already in the buffer—removing it with probability $(1 - p)$ (Lines 3–7). If the number of elements in the buffer reaches the maximal size n, the subsampling operation is executed, which discards each element in χ independently with probability $\frac{1}{2}$; then, p is adjusted to reflect the fact that the buffer now contains each element with probability $p_{\text{new}} = \frac{p_{\text{old}}}{2}$ (Lines 8–10). If the subsampling operation fails, i.e., if no elements get discarded, then the algorithm fails returning $\perp$ (Line 11). After processing the stream, the algorithm returns $\frac{|\chi|}{p}$ as a probably-approximately correct estimate for the number of distinct elements in the stream.

▶ Remark 2. For our discussion below, it is convenient to analyze Algorithm 1 without Line 11, i.e., we will skip the second check of $|\chi| = n$ determining whether the subsampling step succeeded. This modified version simplifies our analysis as we do not have to worry about the possibility of the algorithm failing (returning $\perp$). This transformation is also used in the original CVM proof [10], where the total variational distance between these two variants of the algorithms is shown to be at most $\frac{\delta}{2}$. Thus, probability bounds derived for the modified version can be transferred to the original algorithm, with a correction term of $\frac{\delta}{2}$. ⌟

3.1 Deriving a Simple Probabilistic Invariant

Consider the random variables $X_s := \mathrm{I}(s \in \chi)$ indicating the presence of a stream element $s \in A = \{a_1, \ldots, a_l\}$ in the buffer, where we write I for the indicator of a predicate, so $\mathrm{I}(\text{true}) = 1$ and $\mathrm{I}(\text{false}) = 0$. Before the algorithm first encounters the stream element s, X_s will be 0 unconditionally, because the buffer χ is always a subset of the stream elements processed so far, i.e., $\chi \subseteq \{a_1, \ldots, a_m\}$ after loop iteration m.

In the loop iteration where element s occurs for the first time, it will be inserted with probability p in Lines 3–7. This means, after Line 7, we have:

$$\mathrm{E}[p^{-1} X_s] = 1. \tag{2}$$

Interestingly, this equation is preserved for the rest of the algorithm. For example, let us consider a subsampling step: each s is independently discarded with probability $\frac{1}{2}$ so $\mathcal{P}(X_s = 1)$ is halved, but so is p after subsampling, which preserves the equation.

Let us see how we can verify Equation (2) more formally. For that, we model the state of the randomized algorithm as a pair (χ, p) and we write χ and p for the random variables projecting their respective components from the distribution of the state of the algorithm. We will refer to parts of each loop iteration in Algorithm 1 as step_1 (resp. step_2) for Lines 3–7 (resp. Lines 8–10). The final distribution of the algorithm is the distribution resulting from the sequential composition of alternating steps over the stream:

$$\text{init} \gg\!= \text{step}_1\, a_1 \gg\!= \text{step}_2 \gg\!= \text{step}_1\, a_2 \gg\!= \cdots \gg\!= \text{step}_1\, a_l \gg\!= \text{step}_2$$

where we parameterize step_1 with the stream element that it processes. The term init represents the initial state, i.e., $\text{init} = \textbf{return}\,(\{\}, 1)$. It is easy to show by induction over the sequence of steps, we have $0 < p \leq 1$ and $\chi \subseteq A$ for all possible states of the algorithm.

Let us verify that Equation (2) is preserved as an invariant over all steps. To verify that $\text{step}_1\, a$ preserves Equation (2), we assume some probability space of states Ω fulfills Equation (2) and we would like to show that it is still true for $\Omega \gg\!= \text{step}_1\, a$. By Equation (1),

$$\mathrm{E}_{\Omega \gg\!= \text{step}_1\, a}[p^{-1} X_s] = \int_{\Omega} \int_{\text{Ber}(p)} p^{-1}\, \mathrm{I}\,(s \in (\text{if } \tau \text{ then } \chi \cup \{a\} \text{ else } \chi - \{a\}))\; d\tau d\sigma.$$

Note that we write p or χ even though, we should actually write $p(\sigma)$ or $\chi(\sigma)$, i.e., we remember that these implicitly depend on σ. To see that the right-hand-side is equal to 1, it is useful to consider the cases where $a = s$ and the converse separately. When $a = s$, the right-hand-side is equal to 1 by definition of the Bernoulli distribution (since $p \in (0; 1]$). When $a \neq s$, it follows from the induction hypothesis on Ω; in particular, the term in the inner integral is constant with respect to τ.

The same invariant-based argument is possible for step_2. Let us assume Ω is a probability space of states fulfilling Equation (2). Then by Equation (1),

$$\mathrm{E}_{\Omega \ggg= \mathrm{step}_2}\left[\frac{X_s}{p}\right] = \int_\Omega \left(\text{if } |\chi| = n \text{ then } \left(\int_{\mathrm{subsample}(\chi)} \frac{\mathrm{I}(s \in \tau)}{p/2}\, d\tau\right) \text{ else } \frac{\mathrm{I}(s \in \chi)}{p}\right) d\sigma.$$

Note that the true- and false-cases of the inner if-then-else both evaluate to the same value: $p^{-1}\,\mathrm{I}(s \in \chi)$. If $s \notin \chi$ both sides of the equation are 0, because the subsampling operation returns a subset of χ. If $s \in \chi$ the probability that the element gets subsampled is $1/2$, so we arrive again at $\frac{1/2}{p/2} = p^{-1}\,\mathrm{I}(s \in \chi)$. Hence: $\mathrm{E}_{\Omega \ggg= \mathrm{step}_2}[p^{-1}X_s] = \mathrm{E}_\Omega[p^{-1}X_s] = 1$. This completes the invariance proof for Equation (2).

3.2 Deriving a Functional Probabilistic Invariant

With Equation (2) established, it is straightforward to show that the expected value of the output estimate $p^{-1}|\chi|$ for the modified algorithm (without Line 11) is equal to the desired cardinality $|A|$. However, recall that we are interested in verifying the estimate's PAC guarantee. A typical approach to establishing such a guarantee is to use Chernoff bounds which provide exponential tail bounds (i.e., concentration bounds) for the deviation of sums of independent random variables from their mean. However, these are not directly useful in the CVM algorithm because the key random variables, e.g., $p^{-1}X_s$ for $s \in A$, are dependent.

An alternative is the Cramér–Chernoff method, which is a general method to obtain tail bounds for any random variable. It can be stated simply as $\mathcal{P}(X \geq a) \leq M(t)e^{-ta}$ for all $t > 0$, where $M(t) := \mathrm{E}[\exp(tX)]$ is the moment generating function of the random variable X. It is also possible to obtain lower tail bounds $\mathcal{P}(X \leq a)$ using the Cramér–Chernoff method, which just requires estimates for $M(t)$ for $t < 0$, instead of $t > 0$.

In our case, we are interested in estimating the moment generating function of the random variable $p^{-1}|\chi|$ for the CVM algorithm:

$$\mathrm{E}[\exp(tp^{-1}|\chi|)] = \mathrm{E}\left[\prod_{s \in A} h(p^{-1}X_s)\right]$$

for $h(x) = \exp(tx)$. At this point, it is tempting to see whether the proof for Equation (2) can be generalized to establish bounds for the above. Indeed, we managed to establish the following generalized result:

$$\mathrm{E}\left[\prod_{s \in A} h(p^{-1}X_s)\right] \leq h(1)^{|A|} \tag{3}$$

for every non-negative concave function $h : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$. We call this a *functional probabilistic invariant* because one can establish it for all valid choices of h with a single induction over steps of the randomized algorithm.

Of course, the exponential function in $M(t)$ is convex, so this is not yet the full story. However, we can instead try to derive tail bounds for the random variable $\mathrm{I}(p \geq q)p^{-1}|\chi|$, for some fixed constant $q > 0$. This leads to a similar invariant inequality:

$$\mathrm{E}\left[\prod_{s \in A} I(p \geq q) h(p^{-1} X_s)\right] \leq h(1)^{|A|} \tag{4}$$

with the new condition that h needs to be non-negative and concave only on $[0; q^{-1}]$. This then allows us to approximate the exponential function from above with an affine function h on the range $[0; q^{-1}]$, which yields tail bounds for $p^{-1}|\chi|$ under the condition $p \geq q$. As an example, the upper tail bound can be derived as follows:

$$\begin{aligned}
\mathcal{P}(p^{-1}|\chi| \geq (1+\varepsilon)|A| \wedge p \geq q) &\leq \mathcal{P}(\mathrm{I}(p \geq q) p^{-1}|\chi| \geq (1+\varepsilon)|A|) \\
&\underset{\text{Markov}}{\leq} e^{-t(1+\varepsilon)|A|}\, \mathrm{E}\left[\prod_{s \in A} \mathrm{I}(p \geq q) \exp(t p^{-1} X_s)\right] \\
&\leq e^{-t(1+\varepsilon)|A|}\, \mathrm{E}\left[\prod_{s \in A} \mathrm{I}(p \geq q) h(p^{-1} X_s)\right] \\
&\underset{\text{Ineq. 4}}{\leq} e^{-t(1+\varepsilon)|A|} h(1)^{|A|} \\
&\underset{\text{Calculus}}{\leq} e^{-n\varepsilon^2/12}
\end{aligned}$$

where we choose $h(x) = 1 + qx(e^{t/q} - 1)$. Note that h is affine and it can be easily checked[2] that it is an upper approximation of $\exp(tx)$ for $x \in [0; q^{-1}]$. For the last step, we have to find the t that produces the required bound.[3] To use these bounds, we also have to separately estimate $\mathcal{P}(p < q)$. For that, we use a similar strategy, as in the original proof by Chakraborty et al. [10], with $q = \frac{n}{4|A|}$. The formalization in the supplementary material contains a detailed informal step-by-step proof using our approach in its appendix. Besides the use of Equation (1) and the Cramér–Chernoff method, the steps are elementary.

The main takeaway here is that it is possible to derive useful and general probabilistic invariants by considering expectations of classes of functions of the state, proved using recursion or induction over the algorithm itself. As far as we know this method of establishing tail bounds for randomized algorithms is new.

4 An Unbiased CVM Variant and Negative Dependence

An interesting consequence of our invariant-based approach is that it allowed us to devise and verify a refined version of the CVM algorithm that is both total and unbiased.

4.1 Unbiased CVM Variant

When we look at the subsampling step of Algorithm 1, our invariant inequality (3) imposes the following condition on the subsampling operation:

$$\int_{\mathrm{subsample}(\chi)} \prod_{s \in S} g(\mathrm{I}(s \in \tau))\, d\tau \leq \prod_{s \in S} \mathrm{E}_{\mathrm{Ber}(f)}[g] \tag{5}$$

for all non-negative functions g and any $S \subseteq \chi$. Any subsampling step that satisfies this functional inequality can be used while still preserving inequality (3) for the algorithm.

[2] Because the exponential function is convex and h is affine, we only have to check the end points: $0, q^{-1}$.
[3] We use $t = q \ln(1+\varepsilon)$ which is not the real optimum, but better for algebraic evaluation.

Algorithm 3 New total and unbiased CVM algorithm variant.

Input: Stream elements $a_1, \ldots, a_l$, $0 < \varepsilon$, $0 < \delta < 1$.
Output: A cardinality estimate R for set $A = \{a_1, \ldots, a_l\}$ s.t. $\mathcal{P}\left(|R - |A|| > \varepsilon|A|\right) \leq \delta$

$\chi \leftarrow \{\}, p \leftarrow 1, n = \left\lceil \frac{12}{\varepsilon^2} \ln\left(\frac{3l}{\delta}\right) \right\rceil$, $\frac{1}{2} \leq f < 1$, s.t., nf integer
for $i \leftarrow 1$ to l **do**
  $b \xleftarrow{\$} \mathrm{Ber}(p)$ ▷ insert a_i with probability p (and remove it otherwise)
  if b **then**
    $\chi \leftarrow \chi \cup \{a_i\}$
  else
    $\chi \leftarrow \chi - \{a_i\}$
  if $|\chi| = n$ **then** ▷ if buffer χ is full
    $\chi \xleftarrow{\$} \mathrm{subsample}(\chi)$ ▷ select a random nf-subset of χ
    $p \leftarrow pf$
return $\frac{|\chi|}{p}$ ▷ estimate cardinality of A

Motivated by this observation, our new variant is shown in Algorithm 3. For the subsampling step, instead of keeping each element of χ with probability $\frac{1}{2}$, we instead pick a uniform random nf-subset of χ, where $\frac{1}{2} \leq f < 1$ and such that nf is an integer. For example, it is possible to choose $f = \frac{n-1}{n}$, i.e., discarding a random element from χ in the subsampling step. Since this new subsampling step always reduces the size of χ, the variant is *total* (never returns $\perp$). The invariant-based approach allows us to show that the algorithm is probably-approximately correct and also *unbiased*, i.e., the expectation of the result is exactly $|A|$. These depend crucially on establishing inequality (5) for the new subsampler, for which we need a new concept.

4.2 Background on Negative Dependence

Some sets of random variables possess a property called *negative association*, a generalization of independence. The concept was introduced by Joag-Dev and Proschan [25], who showed that it has many useful closure properties compared to other previously introduced notions of negative dependence, such as negative correlation or negative orthant dependence. Importantly, standard Chernoff–Hoeffding type bounds still apply to negatively associated random variables [14, Prop. 7]. Negative association is defined as follows:

▶ **Definition 3.** *For a function defined on n-tuples $f : V^n \to W$, we will denote by* $\mathrm{dep}(f)$ *the set of coordinates the function depends on, i.e., dep$(f) \subseteq \{1, \ldots, n\}$ is minimal, s.t., $f(x) = f(y)$ for all $x, y \in V^n$ with $x_i = y_i$ for all $i \in$ dep(f).*

▶ **Definition 4** (Negative Association). *A set of random variables $X_1, \ldots, X_n : \Omega \to \mathbb{R}$ is negatively associated if, for all non-decreasing functions $f, g : \mathbb{R}^n \to \mathbb{R}$, which depend on disjoint sets of the variables, i.e.,* $\mathrm{dep}(f) \cap \mathrm{dep}(g) = \emptyset$*, the following inequality holds:*

$$\mathrm{E}[f(X_1, \ldots, X_n) g(X_1, \ldots, X_n)] \leq \mathrm{E}[f(X_1, \ldots, X_n)]\, \mathrm{E}[g(X_1, \ldots, X_n)].$$

The following proposition summarizes some important properties of negatively associated sets of random variables.

▶ **Proposition 5** ([25]). *Summary of results for negatively associated random variables.*

1. *If $X = (X_1, \ldots, X_n)$ are negatively associated then* $\mathrm{E}[f(X)g(X)] \leq \mathrm{E}[f(X)]\, \mathrm{E}[g(X)]$ *for non-increasing functions f, g with* $\mathrm{dep}(f) \cap \mathrm{dep}(g) = \emptyset$.

2. *If $X = (X_1, \dots, X_n)$ are negatively associated, $Y = (Y_1, \dots, Y_m)$ are negatively associated, and the pair of vector-valued random variables X and Y are independent, then the union $X_1, \dots, X_n, Y_1, \dots, Y_m$ is a set of negatively associated random variables.*
3. *If $X = (X_1, \dots, X_n)$ are negatively associated and $f_1, \dots, f_m : \mathbb{R}^n \to \mathbb{R}$ are all non-increasing or all non-decreasing functions, s.t., $\mathrm{dep}(f_i) \cap \mathrm{dep}(f_j) = \emptyset$ for $i \neq j$, then $f_1(X), \dots, f_m(X)$ form a set of negatively dependent random variables of size m.*
4. *If $X_1, \dots, X_n$ are independent then $X_1, \dots, X_n$ are negatively associated.*
5. *A subset of a negatively associated set of random variables is again negatively associated.*

These properties illustrate the trade-off between negative association and independence. For example, Property 3 would be true for independent random variables, even without the condition of monotonicity. To analyze our new subsampler, the following is an important lemma about negative associated random variables.

▶ **Lemma 6.** *Let $X_1, \dots, X_n$ be negatively associated and $f_1, \dots, f_n$ be all non-decreasing (or all non-increasing), non-negative functions, then*

$$\mathrm{E}\left[\prod_{i=1}^{n} f_i(X_i)\right] \leq \prod_{i=1}^{n} f_i(\mathrm{E}[X_i]).$$

Proof. This follows from the definition of negative associativity (or Property 1 of Proposition 5, if the f_i are non-increasing) using induction. ◀

The case for non-decreasing functions of the above lemma is pointed out by Joag-Dev and Proschan [25, P.2]. The reason for our interest in this lemma stems from the fact that indicator variables of random m-subsets are negatively associated. This is a consequence of the fact that permutation distributions are negatively-associated [25, Th. 2.11]. Thus, for the new subsampling step in Line 9 of Algorithm 3, we can derive using Lemma 6:

$$\int_{\mathrm{subsample}(\chi)} \prod_{s \in S} g(\mathrm{I}(s \in \tau))\, d\tau \leq \prod_{s \in S} \int_{\mathrm{subsample}(\chi)} g(\mathrm{I}(s \in \tau))\, d\tau = \prod_{s \in S} \mathrm{E}_{\mathrm{Ber}(f)}[g]. \qquad (6)$$

for any non-negative g and $S \subseteq \chi$. Note that the domain of g has two values, so it is either non-increasing or non-decreasing. Also, if S is a singleton, the inequality becomes an equality. With this ingredient, we can conclude that our results about the original algorithm derived in the previous section also hold for our new variant (Algorithm 3).

5 Formalization of the CVM algorithm

Let us now turn to details of our formalization of the CVM algorithm in Isabelle/HOL using our invariant-based approach. We verified both the total, unbiased variant (Algorithm 3) and the original variant (Algorithm 1) from the introduction.

▶ Note 7. In our supplementary material, the invariant-based verification approach is in the directory `CVM_Invariant`. The theory `CVM_Abstract_Algorithm` verifies a generalized version of the CVM algorithm, with an abstract subsampling operation that is required to fulfill inequality (5). The specialization happens in the following theories, where `CVM_Original_Algorithm` verifies the original algorithm, and `CVM_New_Unbiased_Algorithm` verifies the new total and unbiased variant. Note that only `CVM_New_Unbiased_Algorithm` depends on the new library for negatively associated random variables, which we describe in more detail in Section 6. The total number of lines required for the verification of the

```
context
  fixes f :: real and n :: nat
  assumes f-range: ‹f ∈ {1/2..<1}› ‹n ∗ f ∈ ℕ› and n-gt-0: ‹n > 0›
begin

definition ‹initial-state = State {} 1›                — Setup initial state χ = ∅ and p = 1.
fun subsample where                     — Subsampling operation: Sample random nf subset.
  ‹subsample χ = pmf-of-set {S. S ⊆ χ ∧ card S = n ∗ f}›

fun step where                                                              — Loop body.
  ‹step a (State χ p) = do {
    b ← bernoulli-pmf p;
    let χ = (if b then χ ∪ {a} else χ − {a});

    if card χ = n then do {
      χ ← subsample χ;
      return-pmf (State χ (p ∗ f))
    } else do {
      return-pmf (State χ p)
    }
  }›

fun run-steps where                                    — Iterate loop over stream xs.
  ‹run-steps xs = foldM-pmf step xs initial-state›
fun estimate where ‹estimate (State χ p) = card χ / p›
fun run-algo where                                    — Run algorithm and estimate.
  ‹run-algo xs = map-pmf estimate (run-steps xs)›
[. . .]
end
```

Figure 1 Formalized version of Algorithm 3.

original algorithm is 1003 lines. In addition, we actually verified a slight generalization of Algorithm 1 where the subsampling probability can be any $f \in [\frac{1}{2}; e^{-1/12}]$; the original CVM algorithm [9] is the special case $f = \frac{1}{2}$.

A snippet of the formalization of Algorithm 3 is presented in Figure 1 (the formalization of Algorithm 1 is very similar). We use the same variables as in the informal presentation: n for the maximal size of the buffer, f for the fraction of elements to keep in the buffer when subsampling. The condition ‹$n * f \in \mathbb{N}$› expresses the requirement that the nf must be integer. Instead of representing the state using pairs, as we did in the informal discussion, we use a datatype with the single constructor *State*, which has two arguments χ and p, the buffer and the probability that the stream elements are in the buffer, respectively. Isabelle/HOL provides notation closely related to informal pseudocode, so it is usually feasible to read a formal statement without expert knowledge. Nevertheless, Table 1 contains a brief glossary of the syntax used in the formalization.

The theorem that establishes the correctness of the algorithm, i.e., that the relative error will exceed ε with probability at most δ is expressed in the following snippet:

Table 1 Isabelle/HOL syntax used in Figure 1.

Term	Description
card S	Cardinality of a finite set S.
real	Type of real numbers and conversion from natural numbers into real numbers.
nat	Type of natural numbers (non-negative integers).
bernoulli-pmf p	The probability space over the Boolean values, where the probability of *True* is p. (Bernoulli distribution.)
pmf-of-set S	For a finite set S, the uniform probability space on S. (Every element of S is equiprobable.)
map-pmf f A	The probability space representing the distribution of the random variable f over the probability space A.
return-pmf x	The probability space of the singleton $\{x\}$.
foldM-pmf f xs a	Iterate randomized algorithm f over the sequence xs using the initial state a.

```
theorem correctness:
  assumes ‹ε ∈ {0<..<1::real}› ‹δ ∈ {0<..<1::real}›
  assumes ‹real n ≥ 12 / ε^2 ∗ ln (3 ∗ real (length xs) / δ)›
  defines ‹A ≡ real (card (set xs))›
  shows ‹𝒫(R in run-algo xs. |R − A| > ε ∗ A) ≤ δ›
```

The first line gives conditions on parameters ε and δ, which must be strictly between 0 and 1. The next line requires the buffer size n to be larger than or equal to $12\varepsilon^{-2}\ln(3\delta^{-1}l)$. Then, we introduce the abbreviation A for the cardinality of the set of elements in the sequence *xs*. The notation $\mathcal{P}(x\ in\ M.\ P\ x)$ denotes the probability of a predicate P in the probability space M, so the conclusion gives the PAC guarantee for the output estimate R from *run-algo*.

Similarly, we have also formalized unbiasedness of Algorithm 3:

```
theorem unbiasedness: ‹measure-pmf.expectation (run-algo xs) id = card (set xs)›
```

where the expression *measure-pmf.expectation M f* denotes the expectation of the random variable f on the probability space M.

Our proofs are available both in mechanized form in Isabelle/HOL and as a pen-and-paper proof included in the associated proof document. In practice, we developed the latter proof first and then mechanized it in Isabelle/HOL without much surprises. Most of the lemmas can be one-to-one identified between both proofs; Isabelle's existing libraries, automation capabilities, and structured proof format were used extensively in our proofs.

6 Formalization of a Library for Negative Association

As mentioned in Section 4, formalizing the total and unbiased variant of the CVM algorithm requires results from the theory of negative association.

▶ Note 8. In our supplementary material, the formalization of negative association is in the directory `Neg_Assoc`. This library contains key results used to establish the invariants for CVM (e.g., `Neg_Assoc_Permutation_Distributions`). Although not needed for CVM, we have also mechanized the standard Chernoff bounds (`Neg_Assoc_Chernoff_Bounds`), including the additive bounds by Hoeffding [23, Th. 1, 2] and the multiplicative bounds by Motwani and Raghavan [35, Th. 4.1, 4.2]. Another example application included in the library is proving the false positive rate of Bloom filters (`Neg_Assoc_Bloom_Filters`). In total the library contains 2974 lines of Isabelle code. ⌟

Our formalization follows the definitions by Joag-Dev and Proschan [25] closely. However, their definition leaves the class of test functions f and g (in Definition 4) imprecise. In particular, for the formalization, we limit these functions to those that are bounded and measurable. Additionally, we provide elimination rules, showing that if $X_1, \ldots, X_n$ are negatively associated, then the inequality on expectations is still true even if f, g are only square-integrable; or, alternatively, integrable and non-negative. This is derived using the monotone-convergence theorem.

Another deviation from the original work is that we do not require that the random variables are real-valued. In the formalization, any linearly ordered topological space with the Borel σ-algebra is allowed as the range space. In this case, the test functions must be monotone with respect to the respective order on the range space.

A key issue we faced during formalization, was that there are many theorems that condition on a set of functions being either simultaneously monotone or simultaneously anti-monotone. To reduce duplication, we introduce a notation that allows us to abstract over the direction of relations: $\leq\geq_\eta$; it evaluates to the forward version of the relation $\leq$ if $\eta = \mathit{Fwd}$ and the converse: $\geq$ if $\eta = \mathit{Rev}$. For example the FKG inequality [3, Ch. 6],[18]

$$\mathrm{E}[fg] \geq \mathrm{E}[f]\,\mathrm{E}[g]$$

is true, if f and g are both monotone, or both antimonotone, on a probability space whose domain is a finite distributive lattice with a log-supermodular probability mass function. The reverse inequality is also true, if f is monotone and g is antimonotone, or vice versa. Using our parameterized relation symbol, we can state all variants in a concise manner:

theorem *fkg-inequality-pmf*:
  fixes M :: ‹$('a :: \mathit{finite\text{-}distrib\text{-}lattice})\ \mathit{pmf}$›
  fixes $f\ g$ :: ‹$'a \Rightarrow \mathit{real}$›
  assumes ‹$\bigwedge x\ y.\ \mathit{pmf}\ M\ x * \mathit{pmf}\ M\ y \leq \mathit{pmf}\ M\ (x \sqcup y) * \mathit{pmf}\ M\ (x \sqcap y)$›
  assumes ‹$\mathit{monotone}\ (\leq)\ (\leq\geq_\tau)\ f$› ‹$\mathit{monotone}\ (\leq)\ (\leq\geq_\sigma)\ g$›
  shows ‹$(\int x.\ f\ x\ \partial M) * (\int x.\ g\ x\ \partial M) \leq\geq_{\tau\, *\, \sigma}\ (\int x.\ f\ x * g\ x\ \partial M)$›

Here, σ and τ are relation directions, and $\sigma * \tau$ multiplies relation directions, i.e., $\sigma * \tau$ is the forward direction if σ and τ have the same direction, and it is the reverse direction otherwise. The first assumption denotes the log-supermodularity of the probability mass function, while the second assumptions are the parametric monotonicity conditions. The FKG inequality is a key result which enables verification of negative association for random variables. This includes the indicator variables for the new subsampling operation we introduced in Section 4.

Let us summarize a few key formalized results for negatively associated random variables in our library. The following is the well-known Hoeffding inequality [23] generalized for negatively associated random variables.

lemma *hoeffding-bound-two-sided*:
  assumes ‹$\mathit{neg\text{-}assoc}\ X\ I$› ‹$\mathit{finite}\ I$›
  assumes ‹$\bigwedge i.\ i \in I \Longrightarrow a\ i \leq b\ i$›
  assumes ‹$\bigwedge i.\ i \in I \Longrightarrow \mathit{AE}\ \omega\ \mathit{in}\ M.\ X\ i\ \omega \in \{a\ i..b\ i\}$› ‹$I \neq \{\}$›
  defines ‹$n \equiv \mathit{real}\ (\mathit{card}\ I)$›
  defines ‹$\mu \equiv (\sum i \in I.\ \mathit{expectation}\ (X\ i))$›
  assumes ‹$\delta \geq 0$› ‹$(\sum i \in I.\ (b\ i - a\ i)^2) > 0$›
  shows ‹$\mathcal{P}(\omega\ \mathit{in}\ M.\ |(\sum i \in I.\ X\ i\ \omega) - \mu| \geq \delta * n) \leq 2 * \mathit{exp}\ (-2 * (n * \delta)^2\ /\ (\sum i \in I.\ (b\ i - a\ i)^2))$›

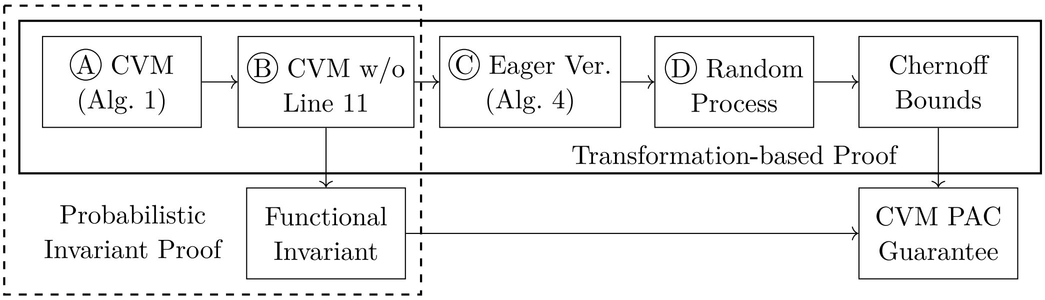

Figure 2 An overview of the two formalized proof approaches for the CVM algorithm.

Another key result (shown below) used for the verification of our CVM variant is the negative-associativity of the indicator functions of random k-subsets of a finite set S (with cardinality greater than or equal to k).

lemma *n-subsets-distribution-neg-assoc*:
  assumes ‹*finite* S› ‹$k \leq$ *card* S›
  defines ‹$p \equiv$ *pmf-of-set* $\{T.\ T \subseteq S \wedge$ *card* $T = k\}$›
  shows ‹*measure-pmf.neg-assoc* p $(\in)$ S›

This is a consequence of a more general result, which we have also shown, that permutation distributions are negatively associated. We relied on the proof by Dubashi using the FKG inequality [13, Th. 10]; there is a prior proof by Joag-Dev and Proschan [25, Th. 2.11], which seemed to be incorrect.

7 Transformation-Based Proof

Here, we describe the transformation-based proof by Chakraborty et al. [10], focusing on the challenging parts in its formalization. An overview of the proof is shown in Figure 2, which highlights, in part, why the transformation-based approach required more work to formalize, and why we developed our new approach using probabilistic invariants.

▶ Note 9. In our supplementary material, the transformation-based formalization of CVM is in the directory `CVM_Transforms`. To formalize probabilistic transformations (relating two distributions), we built on an existing relational program logic in Isabelle/HOL [31]. The formalization took 2630 lines which is considerably longer than the proof using our invariant-based technique (1003 lines). ⌟

As mentioned in Section 1, the main difficulty in directly analyzing Algorithm 1 is the lack of independence in its state variables. The technique Chakraborty et al. use to circumvent this issue is by progressively modifying the algorithm (A–D in Figure 2), in a manner that obviously bounds (or preserves) its distribution, and such that the final algorithm D can be described using a simple random process with independent coin flips.

For interested readers, A corresponds to [10, Algorithm 1], B is [10, Algorithm 2], and D is [10, Algorithm 3]. Whereas Chakraborty et al. move directly from B to D with an informal argument, C is a transformation we added in the formalization to bridge this gap.

7.1 A Bridging Transformation

Let us consider algorithm B in a state where k subsampling steps have been performed, i.e., $p = 2^{-k}$. The algorithm would perform a coin flip lazily with probability p when it encounters

Algorithm 4 Modified CVM algorithm with independent coin flips. The function last_index returns the index of the last occurrence of an element in the sequence, before the current iteration.

Input: Stream elements $a_1, \dots, a_l$, $0 < \varepsilon$, $0 < \delta < 1$.
Output: A cardinality estimate R for set $A = \{a_1, \dots, a_l\}$ s.t. $\mathcal{P}\left(|R - |A|| > \varepsilon|A|\right) \le \delta$

$\chi \leftarrow \{\}, k \leftarrow 0, n = \left\lceil \frac{12}{\varepsilon^2} \ln\left(\frac{6l}{\delta}\right) \right\rceil$
$b[i,j] \xleftarrow{\$} \mathrm{Ber}(1/2)$ for $i, j \in \{1, \cdots, l\}$ ▷ perform l^2 unbiased independent coin flips
for $i \leftarrow 1$ to l **do**
  if $b[i,1] = b[i,2] = \cdots = b[i,k] = 1$ **then** ▷ insert a_i if first k flips are 1s.
    $\chi \leftarrow \chi \cup \{a_i\}$
  else
    $\chi \leftarrow \chi - \{a_i\}$
  if $|\chi| = n$ **then** ▷ if buffer χ is full
    $\chi \leftarrow \{a \in \chi \mid b[\mathrm{last\_index}(a), k+1] = 1\}$ ▷ keep elems. whose $k+1$-th flip is 1
    $k \leftarrow k + 1$
return $2^k|\chi|$ ▷ estimate cardinality of A

the next stream element. The transformation C is shown in Algorithm 4, and we prove that it computes precisely the same distribution as B. In C, we eagerly perform a fixed number of coin flips for each sequence element at the beginning. Now, each element is put into the state χ, whenever the first k coin flips associated with the sequence element are all 1s. This happens exactly with probability 2^{-k}, which means the behaviour of the algorithm is unchanged from B. Similarly, in the subsampling operation, only those elements whose $k+1$-th associated coin flip is 1 are kept; the operation $p \mapsto \frac{p}{2}$ is replaced with $k \mapsto k+1$. This again preserves the behaviour of B that each element is discarded independently with probability $1/2$.

It is easy to show for C that the coin flips are independent, and that the set of elements in χ in any state are exactly those stream elements for which the first k entries of their associated coin flips are 1. The final random process D directly computes the final set of elements in χ after the stream, taking K as a fixed parameter; one relates C to D by:

$$\mathcal{P}_{\mathrm{C}}(k = K \wedge \chi = X) \le \mathcal{P}_{\mathrm{D}_K}(\chi = X)$$

for fixed values of K and X. To see how tail bounds can be derived from this inequality, let us first consider the failure event where the algorithm C's estimate exceeds the desired estimation interval and it ends with some fixed value $k = K$. Using D, this can be bounded using a Chernoff bound for the probability that the number of stream elements whose associated coin flips start with K 1s is outside $2^{-K}|A|(1 \pm \varepsilon)$. Now, we can take a union bound over all the possible values K to establish a global bound for the failure event in C. This is explained in more detail by Chakraborty et al. [10].

7.2 Eager to Lazy Coin Flips

A remaining question is how to formalize the transformation from B to C. Our insight is that it is best to solve the problem backwards, i.e., we start with the modified algorithm C, which performs all the coin flips in advance *eagerly* and convert it back to B which implicitly performs the coin flips *lazily* at the point they are needed.

The main idea is to automatically push down the coin flips through the expression tree of Algorithm 4. To explain how this works, let us first define the *sampling* function, i.e., let f

be a function that takes as argument a vector of coin flips indexed by I, then we can express the distribution of f with respect to independent unbiased coin flips as:

$$\textit{sample } f = \textit{map-pmf } f \ (\textit{prod-pmf } I \ (\lambda\text{-}.\ \textit{bernoulli-pmf } (1/2)))$$

The interesting fact is that we can distribute the sampling operation over composition:

▶ **Observation 10.** *Let f, g be functions consuming a set of coin flips (indexed by I), where g also consumes the output of f, such that, f depends only on the coin flips indexed by $J \subseteq I$ and g depends on the complement $I - J$, then:*

$$\textit{sample } (\lambda\omega.\ g\ \omega \circ f\ \omega) = \textit{sample } f \ggg (\lambda x.\ \textit{sample } (\lambda\omega.\ g\ \omega\ x))$$

By recursively applying the observation, we end up with elementary lookup operations, e.g., *sample* $(\lambda\omega.\ \omega\ i)$, for which it is easy to see that it is just a coin flip, i.e., equal to *bernoulli-pmf* $(1/2)$. This lets us readily transform Ⓒ to Ⓑ and prove their distributions equivalent.

A detail that we have simplified here is that the split of the index sets, e.g., which coin flips f depends on and which coin flips g depends on, may be dynamic. For example, when the algorithm increases the subsampling counter k, it will have read the corresponding row of coin flips. This means we have a situation where the previous loop iteration communicates to the next loop iteration which coin flips it depends on using the state. And the next loop iteration will indeed only read coin flips that were not read by the previous iteration.

To handle these situations we generalized Observation 10 to allow for the case where the cut between the set of coin flips f and g depend on, may depend on the result of f.

8 Related Work

8.1 Algorithms for the Distinct Elements Problem

It is important to note that there are several practical solutions for the distinct elements problem. The first solution was presented by Flajolet [17] in 1985; however, like many other authors [16, 22, 38], his solution makes the assumption that a fixed hash function can be regarded as a fully random function. Alon et al. [2, Section 2.3] presented an easy remedy, which does not require such unmotivated model assumptions. Their algorithm just relies on keeping track of the maximum of the hash values of the stream elements, where the hash function must be chosen uniformly from a pairwise independent family; the space complexity of this algorithm is $\mathcal{O}(\varepsilon^{-2}\ln(\delta^{-1})b)$, where we assume that b is the number of bits required to represent the stream elements. This is slightly better than the CVM algorithm which requires $\mathcal{O}(\varepsilon^{-2}\ln(\delta^{-1}l)b)$ since there is no logarithmic dependency on l.

Later, Bar-Yossef et al. [4], Kane et al. [26] and Błasiok in 2020 [7] presented increasingly sophisticated solutions. The last one by Błasiok is optimal, with a space complexity of $\mathcal{O}(\varepsilon^{-2}\ln(\delta^{-1}) + b)$. Karayel [28] presented a version of the latter that preserves monotonicity and supports the merge-operation, which enables its use in distributed settings, such as Map-Reduce pipelines [12]. It should be noted that these recent algorithms are mostly of theoretical interest, as the constants, as well as the implementation complexity, are rather large. What makes the CVM algorithm unique is its simplicity and the fact that it does not rely on hashing, which may enable more general use-cases than the traditional algorithms.

The aforementioned hash-based algorithms are biased; Flajolet et al. [17] points this out and also provides bounds on the distance between the expected result and the cardinality of the stream. Most authors do not discuss the matter of biasedness but it is not hard to show. One issue, for example, is that the usual method to amplify the accuracy of these algorithms is using the median, which does not preserve expectations. In the context of query processing, unbiasedness has been discussed [21, Section 2.1], but we could not find any similar discussion for the distinct elements problem in the streaming model.

8.2 Probabilistic Invariants and Formalization

As far as we know, probabilistic invariants have not been used to establish Gaussian tail-bounds. However, it is fairly common to use recursive analysis techniques to establish results about expectations or variance of random variables, such as their run-time [35, Section 1.4]. This is easy due to the linearity of expectations and—for independent random variables—linearity of variances. A simple example is the Morris-counter [39] or the expected run-time of the quick-sort algorithm [33, Section 2.5].

There is also research on the (automated) analysis of loop invariants, for probabilistic loops, using their characteristic functions [6, 32]. This approach works by establishing the limiting distribution of the state of the loop. De Medeiros et al. [11, Section 3.2] also establish methods to derive limiting distributions of probabilistic loops. Our approach differs from these techniques, by avoiding computation of the distribution, which, we think, is infeasible for the CVM algorithm. Instead, we investigate invariants of classes of functions of the distributions, which are relevant for the analysis. There is research on automated evaluation of moments for restricted classes of loops, which contain only polynomial assignments and no branches [5, 30]. However, these methods do not extend to algorithms with branches or, more generally, algorithms which contain discrete operations.

Finally, verification of randomized algorithms has been tackled by various authors using various proof assistants [8, 11, 15, 20, 24, 40, 41]. The most closely related efforts are the mechanizations of frequency moments algorithms by Karayel [27, 28]. The functional invariant proof technique we introduce here should be applicable in any higher-order setting.

9 Conclusion

We presented the first formalization of the CVM algorithm using Isabelle/HOL. Central to our formalization is a novel invariant-based proof technique for randomized algorithms, which is inspired by our alternative analysis of the CVM algorithm via the Cramér–Chernoff method; comparing our approach against the original proof by Chakraborty et al. [10] shows that our technique yields a considerably shorter formalization (with 1003 vs. 2630 lines). Interestingly, our technique also readily generalized to a new CVM variant with stronger properties (totality and unbiasedness)—we formalized this latter version using the same invariant, together with a new library of results for negative association.

In future work, it would be interesting to formalize other variations of subsampling for CVM. More generally, one could further explore whether the technique we introduced here could be applied towards proofs of other randomized (streaming) algorithms.

References

1 Archive of Formal Proofs. Accessed: 2025-02-26. URL: `https://isa-afp.org`.

2 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999. doi:10.1006/JCSS.1997.1545.

3 Noga Alon and Joel H. Spencer. *The Probabilistic Method, Second Edition.* John Wiley, 2000. doi:10.1002/0471722154.

4. Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In José D. P. Rolim and Salil P. Vadhan, editors, *RANDOM*, volume 2483 of *LNCS*, pages 1–10. Springer, 2002. `doi:10.1007/3-540-45726-7_1`.
5. Ezio Bartocci, Laura Kovács, and Miroslav Stankovič. Automatic generation of moment-based invariants for Prob-solvable loops. In Yu-Fang Chen, Chih-Hong Cheng, and Javier Esparza, editors, *ATVA*, volume 11781 of *LNCS*, pages 255–276. Springer, 2019. `doi:10.1007/978-3-030-31784-3_15`.
6. Kevin Batz, Mingshuai Chen, Sebastian Junges, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. Probabilistic program verification via inductive synthesis of inductive invariants. In Sriram Sankaranarayanan and Natasha Sharygina, editors, *TACAS*, volume 13994 of *LNCS*, pages 410–429. Springer, 2023. `doi:10.1007/978-3-031-30820-8_25`.
7. Jarosław Błasiok. Optimal streaming and tracking distinct elements with high probability. *ACM Trans. Algorithms*, 16(1):3:1–3:28, 2020. `doi:10.1145/3309193`.
8. Azucena Garvía Bosshard, Jonathan Bootle, and Christoph Sprenger. Formal verification of the Sumcheck protocol. In *CSF*, pages 605–619. IEEE, 2024. `doi:10.1109/CSF61375.2024.00014`.
9. Sourav Chakraborty, N. V. Vinodchandran, and Kuldeep S. Meel. Distinct elements in streams: An algorithm for the (text) book. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *ESA*, volume 244 of *LIPIcs*, pages 34:1–34:6. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/LIPICS.ESA.2022.34`.
10. Sourav Chakraborty, N. V. Vinodchandran, and Kuldeep S. Meel. Distinct elements in streams: An algorithm for the (text) book. *CoRR*, abs/2301.10191, 2023. `doi:10.48550/ARXIV.2301.10191`.
11. Markus de Medeiros, Muhammad Naveed, Tancrède Lepoint, Temesghen Kahsai, Tristan Ravitch, Stefan Zetzsche, Anjali Joshi, Joseph Tassarotti, Aws Albarghouthi, and Jean-Baptiste Tristan. Verified foundations for differential privacy. *CoRR*, abs/2412.01671, 2024. `doi:10.48550/ARXIV.2412.01671`.
12. Jeffrey Dean and Sanjay Ghemawat. MapReduce: a flexible data processing tool. *Commun. ACM*, 53(1):72–77, 2010. `doi:10.1145/1629175.1629198`.
13. Devdatt P. Dubhashi, Volker Priebe, and Desh Ranjan. Negative dependence through the FKG inequality. *BRICS Report Series*, 3(27), 1996. `doi:10.7146/brics.v3i27.20008`.
14. Devdatt P. Dubhashi and Desh Ranjan. Balls and bins: A study in negative dependence. *Random Struct. Algorithms*, 13(2):99–124, 1998. `doi:10.1002/(SICI)1098-2418(199809)13:2<99::AID-RSA1>3.0.CO;2-M`.
15. Manuel Eberl, Max W. Haslbeck, and Tobias Nipkow. Verified analysis of random binary tree structures. *J. Autom. Reason.*, 64(5):879–910, 2020. `doi:10.1007/S10817-020-09545-0`.
16. Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. In *Conference on Analysis of Algorithms*, 2007. `doi:10.46298/dmtcs.3545`.
17. Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985. `doi:10.1016/0022-0000(85)90041-8`.
18. C. M. Fortuin, P. W. Kasteleyn, and J. Ginibre. Correlation inequalities on some partially ordered sets. *Commun. Math. Phys.*, 22(2):89–103, 1971. `doi:10.1007/BF01651330`.
19. Michèle Giry. A categorical approach to probability theory. In B. Banaschewski, editor, *Categorical Aspects of Topology and Analysis*, volume 915 of *LNM*, pages 68–85. Springer, 1982. `doi:10.1007/BFb0092872`.
20. Kiran Gopinathan and Ilya Sergey. Certifying certainty and uncertainty in approximate membership query structures. In Shuvendu K. Lahiri and Chao Wang, editors, *CAV*, volume 12225 of *LNCS*, pages 279–303. Springer, 2020. `doi:10.1007/978-3-030-53291-8_16`.
21. Peter J. Haas, Jeffrey F. Naughton, S. Seshadri, and Lynne Stokes. Sampling-based estimation of the number of distinct values of an attribute. In Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio, editors, *VLDB*, pages 311–322. Morgan Kaufmann, 1995.

22 Stefan Heule, Marc Nunkesser, and Alexander Hall. HyperLogLog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm. In Giovanna Guerrini and Norman W. Paton, editors, *EDBT*, pages 683–692. ACM, 2013. doi:10.1145/2452376.2452456.

23 Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963. doi:10.2307/2282952.

24 Joe Hurd. Formal verification of probabilistic algorithms. Technical Report UCAM-CL-TR-566, University of Cambridge, Computer Laboratory, 2003. doi:10.48456/tr-566.

25 Kumar Joag-Dev and Frank Proschan. Negative association of random variables with applications. *Annals of Statistics*, 11:286–295, 1983. doi:10.1214/AOS/1176346079.

26 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In Jan Paredaens and Dirk Van Gucht, editors, *PODS*, pages 41–52. ACM, 2010. doi:10.1145/1807085.1807094.

27 Emin Karayel. Formalization of randomized approximation algorithms for frequency moments. In June Andronick and Leonardo de Moura, editors, *ITP*, volume 237 of *LIPIcs*, pages 21:1–21:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ITP.2022.21.

28 Emin Karayel. An embarrassingly parallel optimal-space cardinality estimation algorithm. In Nicole Megow and Adam D. Smith, editors, *APPROX/RANDOM*, volume 275 of *LIPIcs*, pages 35:1–35:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.APPROX/RANDOM.2023.35.

29 Donald E. Knuth. The CVM algorithm for estimating distinct elements in streams, 2023. Accessed: 2025-01-14. URL: https://www-cs-faculty.stanford.edu/~knuth/papers/cvm-note.pdf.

30 Andrey Kofnov, Marcel Moosbrugger, Miroslav Stankovic, Ezio Bartocci, and Efstathia Bura. Moment-based invariants for probabilistic loops with non-polynomial assignments. In Erika Ábrahám and Marco Paolieri, editors, *QEST*, volume 13479 of *LNCS*, pages 3–25. Springer, 2022. doi:10.1007/978-3-031-16336-4_1.

31 Andreas Lochbihler. Probabilistic functions and cryptographic oracles in higher order logic. In Peter Thiemann, editor, *ESOP*, volume 9632 of *LNCS*, pages 503–531. Springer, 2016. doi:10.1007/978-3-662-49498-1_20.

32 Annabelle McIver and Carroll Morgan. *Probabilistic loops: Invariants and variants*, pages 37–78. Springer, New York, NY, 2005. doi:10.1007/0-387-27006-X_2.

33 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2005. doi:10.1017/CBO9780511813603.

34 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis.* Cambridge University Press, USA, 2nd edition, 2017.

35 Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995. doi:10.1017/CBO9780511814075.

36 Steve Nadis. Computer scientists invent an efficient new way to count, 2024. Accessed: 2025-01-14. URL: https://www.quantamagazine.org/computer-scientists-invent-an-efficient-new-way-to-count-20240516.

37 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002. doi:10.1007/3-540-45949-9.

38 Seth Pettie and Dingyu Wang. Information theoretic limits of cardinality estimation: Fisher meets Shannon. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC*, pages 556–569. ACM, 2021. doi:10.1145/3406325.3451032.

39 Robert H. Morris Sr. Counting large numbers of events in small registers. *Commun. ACM*, 21(10):840–842, 1978. doi:10.1145/359619.359627.

40 Daniel Stüwe and Manuel Eberl. Probabilistic primality testing. *Archive of Formal Proofs*, 2019. https://isa-afp.org/entries/Probabilistic_Prime_Tests.html, Formal proof development.

41 Yong Kiam Tan, Jiong Yang, Mate Soos, Magnus O. Myreen, and Kuldeep S. Meel. Formally certified approximate model counting. In Arie Gurfinkel and Vijay Ganesh, editors, *CAV*, volume 14681 of *LNCS*, pages 153–177. Springer, 2024. doi:10.1007/978-3-031-65627-9_8.