Verification of the CVM algorithm with a New Recursive Analysis Technique

Emin Karayel, Derek Khu, Kuldeep S. Meel, Yong Kiam Tan, and Seng Joe Watt

March 19, 2025

Abstract

In 2022, Chakraborty et al. [1] published a streaming algorithm (henceforth, the CVM algorithm) for the distinct elements problem, that deviated considerably from the state-of-the art, due to its simplicity and avoidance of standard derandomization techniques, while still maintaining a close to optimal logarithmic space complexity.

In this entry, we verify the CVM algorithm's correctness using a new technique which simplifies the analysis considerably compared to the orignal proof by Chakraborty et al. The main idea is based on a probabilistic invariant that allows us to derive concentration bounds using the Cramér–Chernoff method.

This new technique opens up the possible algorithm design space, and we introduce a new variant of the CVM algorithm, that is total, and also has an additional property in addition to concentration: unbiasedness. This means the expected result of the algorithm is exactly equal to the desired result. The latter is also a new property, that neither the original CVM algorithm nor classic algorithms for the distinct elements problem possess.

Contents

1 Preliminary Definitions and Results

theory *CVM-Preliminary*
  imports *HOL−Probability.SPMF*
begin

lemma *bounded-finite*:
  assumes ‹*finite S*›
  shows ‹*bounded* (*f* ‘ *S*)›
  ⟨*proof*⟩

lemma *of-bool-power*:
  assumes ‹$y > 0$›
  shows ‹(*of-bool x*::*real*) ^(*y*::*nat*) = *of-bool x*›
  ⟨*proof*⟩

lemma *card-filter-mono*:
  assumes ‹*finite S*›
  shows ‹*card* (*Set.filter p S*) ≤ *card S*›
  ⟨*proof*⟩

fun *foldM* ::
  ‹('a ⇒ ('b ⇒ 'c) ⇒ 'c) ⇒ ('b ⇒ 'c) ⇒ ('d ⇒ 'b ⇒ 'a) ⇒ 'd list ⇒ 'b ⇒ 'c› **where**
  ‹*foldM* - *return'* - [] *val* = *return' val*› |
  ‹*foldM bind' return' f* (*x* # *xs*) *val* =
    bind' (*f x val*) (*foldM bind' return' f xs*)›

abbreviation *foldM-pmf* ::
  ‹('a ⇒ 'b ⇒ 'b pmf) ⇒ 'a list ⇒ 'b ⇒ 'b pmf› **where**
  ‹*foldM-pmf* ≡ *foldM bind-pmf return-pmf*›

lemma *foldM-pmf-snoc*: ‹*foldM-pmf f* (*xs*@[*y*]) *val* = *bind-pmf* (*foldM-pmf f xs val*) (*f y*)›
  ⟨*proof*⟩

abbreviation *foldM-spmf*
  :: ‹('a ⇒ 'b ⇒ 'b spmf) ⇒ 'a list ⇒ 'b ⇒ 'b spmf› **where**
  ‹*foldM-spmf* ≡ *foldM bind-spmf return-spmf*›

lemma *foldM-spmf-snoc*: ‹*foldM-spmf f* (*xs*@[*y*]) *val* = *bind-spmf* (*foldM-spmf f xs val*) (*f* *y*)›
  ⟨*proof*⟩

abbreviation ‹*prob-fail* ≡ (λ*x*. *pmf x None*)›

abbreviation ‹*fail-spmf* ≡ *return-pmf None*›

abbreviation *fails-or-satisfies* :: ‹('a ⇒ bool) ⇒ 'a option ⇒ bool› **where**
  ‹*fails-or-satisfies* ≡ *case-option True*›

lemma *prob-fail-foldM-spmf-le* :
  fixes
    p :: *real* **and**
    P :: ‹'b ⇒ *bool*› **and**

f :: ‹$'a \Rightarrow 'b \Rightarrow 'b$ *spmf*›
assumes
‹$\bigwedge$ x y z. P y $\Longrightarrow$ z $\in$ *set-spmf* (f x y) $\Longrightarrow$ P z›
‹$\bigwedge$ x *val*. P *val* $\Longrightarrow$ *prob-fail* (f x *val*) $\leq$ p›
‹P *val*›
shows ‹*prob-fail* (*foldM-spmf* f *xs val*) $\leq$ *real* (*length xs*) $*$ p›
⟨*proof*⟩

lemma *foldM-spmf-of-pmf-eq* :
shows ‹*foldM-spmf* (λx y. *spmf-of-pmf* (f x y)) *xs* = *spmf-of-pmf* $\circ$ *foldM-pmf* f *xs*›
(**is** *?thesis-0*)
and ‹*foldM-spmf* (λx y. *spmf-of-pmf* (f x y)) *xs val* = *spmf-of-pmf* (*foldM-pmf* f *xs val*)›
(**is** *?thesis-1*)
⟨*proof*⟩

end

2 Abstract Algorithm

This section verifies an abstract version of the CVM algorithm, where the subsampling step can be an arbitrary randomized algorithm fulfilling an expectation invariant.

The abstract algorithm is presented in Algorithm 1.

Algorithm 1 Abstract CVM algorithm.

Input: Stream elements $a_1, \ldots, a_l$, $0 < \varepsilon$, $0 < \delta < 1$, $1/2 \leq f < 1$
Output: An estimate R, s.t., $\mathcal{P}\left(|R - |A|| > \varepsilon |A|\right) \leq \delta$ where $A := \{a_1, \ldots, a_l\}$.

$\chi \leftarrow \{\}, p \leftarrow 1, n \geq \left\lceil \frac{12}{\varepsilon^2} \ln(\frac{3l}{\delta}) \right\rceil$
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**
        $\chi \xleftarrow{\$} \mathrm{subsample}(\chi)$ ▷ abstract subsampling step
        $p \leftarrow pf$
return $\frac{|\chi|}{p}$ ▷ estimate cardinality of A

For the subsampling step we assume that it fulfills the following inequality:

$$\int_{\mathrm{subsample}(\chi)} \left(\prod_{i \in S} g(i \in \omega) \right) d\omega \leq \prod_{i \in S} \left(\int_{Ber(f)} g(\omega)\, d\omega \right) \qquad (1)$$

for all non-negative functions g and $S \subseteq \chi$, where $\mathrm{Ber}(p)$ denotes the Bernoulli-distribution.

The original CVM algorithm uses a subsampling step where each element of χ is retained independently with probability f. It is straightforward to see that this fulfills the above condition (with equality).

The new CVM algorithm variant proposed in this work uses a subsampling step where a random nf-sized subset of χ is kept. This also fulfills the above inequality, although this is harder to prove and will be explained in more detail in Section 4.

In this section, we will verify that the above abstract algorithm indeed fulfills the desired conditions on its estimate, as well as unbiasedness, i.e., that: $\mathbb{E}[R] = |A|$. The part that is not going to be verified in this section, is the fact that the algorithm keeps at most n elements in the state χ, because it is not unconditionally true, but will be ensured (by different means) for the concrete instantiations in the following sections.

An informal version of this proof is presented in Appendix A. For important lemmas and theorems, we include a reference to the corresponding statement in the appendix.

theory *CVM-Abstract-Algorithm*

imports
  HOL−Decision-Procs.Approximation
  CVM-Preliminary
  Finite-Fields.Finite-Fields-More-PMF
  Universal-Hash-Families.Universal-Hash-Families-More-Product-PMF
begin

unbundle *no vec-syntax*

datatype *'a state* = *State* (*state-χ*: ‹*'a set*›) (*state-p*: *real*)

datatype *'a run-state* = *FinalState* ‹*'a list*› | *IntermState* ‹*'a list*› ‹*'a*›

lemma *run-state-induct*:
  assumes ‹*P* (*FinalState* [])›
  assumes ‹$\bigwedge$*xs x*. *P* (*FinalState xs*) ⟹ *P* (*IntermState xs x*)›
  assumes ‹$\bigwedge$*xs x*. *P* (*IntermState xs x*) ⟹ *P* (*FinalState* (*xs*@[*x*]))›
  shows ‹*P result*›
⟨*proof*⟩

locale *cvm-algo-abstract* =
  fixes *n* :: *nat* **and** *f* :: *real* **and** *subsample* :: ‹*'a set* ⇒ *'a set pmf*›
  assumes *n-gt-0*: ‹*n* > *0*›
  assumes *f-range*: ‹*f* ∈ {*1/2..<1*}›
  assumes *subsample*:
    ‹$\bigwedge$*U x*. *card U* = *n* ⟹ *x* ∈ *set-pmf* (*subsample U*) ⟹ *x* ⊆ *U*›
  assumes *subsample-inequality*:
    ‹$\bigwedge$*g U S*. *S* ⊆ *U*
      ⟹ *card U* = *n*
      ⟹ *range g* ⊆ {*0::real..*}
      ⟹ ($\int$ω. ($\prod$*s*∈*S*. *g*(*s* ∈ ω)) ∂*subsample U*) ≤ ($\prod$*s*∈*S*. ($\int$ω. *g* ω ∂*bernoulli-pmf f*))›
begin

Line 1 of Algorithm 1:

definition *initial-state* :: ‹*'a state*› **where**
  ‹*initial-state* ≡ *State* {} *1*›

Lines 3–7:

fun *step-1* :: ‹*'a* ⇒ *'a state* ⇒ *'a state pmf*› **where**
  ‹*step-1 a* (*State* χ *p*) =
    do {
      b ← *bernoulli-pmf p*;
      let χ = (*if b then* χ ∪ {*a*} *else* χ − {*a*});

      return-pmf (*State* χ *p*)
    }›

Lines 8–10:

fun *step-2* :: ‹*'a state* ⇒ *'a state pmf*› **where**
  ‹*step-2* (*State* χ *p*) = *do* {
    if card χ = *n*
    then do {

$\chi \leftarrow$ *subsample* χ;
return-pmf (*State* χ $(p*f)$)
} *else do* {
return-pmf (*State* χ p)
}
}›

schematic-goal *step-1-def*: ‹*step-1 x σ = ?x*›
⟨*proof*⟩

schematic-goal *step-2-def*: ‹*step-2 σ = ?x*›
⟨*proof*⟩

Lines 1–10:

definition *run-steps* :: ‹*'a list ⇒ 'a state pmf*› **where**
‹*run-steps xs ≡ foldM-pmf* ($\lambda x\ \sigma$. *step-1 x σ* ≫= *step-2*) *xs initial-state*›

Line 11:

definition *estimate* :: ‹*'a state ⇒ real*› **where**
‹*estimate σ = card* (*state-χ σ*) / *state-p σ*›

lemma *run-steps-snoc*: ‹*run-steps* (*xs* @[*x*]) = *run-steps xs* ≫= *step-1 x* ≫= *step-2*›
⟨*proof*⟩

fun *run-state-pmf* **where**
‹*run-state-pmf* (*FinalState xs*) = *run-steps xs*› |
‹*run-state-pmf* (*IntermState xs x*) = *run-steps xs* ≫= *step-1 x*›

fun *len-run-state* **where**
‹*len-run-state* (*FinalState xs*) = *length xs*› |
‹*len-run-state* (*IntermState xs x*) = *length xs*›

fun *run-state-set* **where**
‹*run-state-set* (*FinalState xs*) = *set xs*› |
‹*run-state-set* (*IntermState xs x*) = *set xs* ∪ {*x*}›

lemma *finite-run-state-set*[*simp*]: ‹*finite* (*run-state-set σ*)› ⟨*proof*⟩

lemma *subsample-finite-pmf*:
assumes ‹*card U = n*›
shows ‹*finite* (*set-pmf* (*subsample U*))›
⟨*proof*⟩

lemma *finite-run-state-pmf*: ‹*finite* (*set-pmf* (*run-state-pmf ϱ*))›
⟨*proof*⟩

lemma *state-χ-run-state-pmf*: ‹*AE σ in run-state-pmf ϱ. state-χ σ* ⊆ *run-state-set ϱ*›
⟨*proof*⟩

lemma *state-χ-finite*: ‹*AE σ in run-state-pmf ϱ. finite* (*state-χ σ*)›
⟨*proof*⟩

lemma *state-p-range*: ‹*AE σ in run-state-pmf ϱ. state-p σ* ∈ {*0*<..*1*}›

⟨*proof*⟩

Lemma 1:

lemma *run-steps-preserves-expectation-le*:
  fixes φ :: ‹*real* ⇒ *bool* ⇒ *real*›
  assumes *phi* :
    ‹$\bigwedge$ *x b*. ⟦*0* < *x*; *x* ≤ *1*⟧ ⟹ φ *x b* ≥ *0*›
    ‹$\bigwedge$ *p x*. ⟦*0* < *p*; *p* ≤ *1*; *0* < *x*; *x* ≤ *1*⟧ ⟹ ($\int$ ω. φ *x* ω ∂*bernoulli-pmf p*) ≤ φ (*x* / *p*)
True›
    ‹*mono-on* {*0..1*} (λ*x*. φ *x False*)›
  defines ‹*aux* ≡ λ *S* σ. ($\prod$ *x* ∈ *S*. φ (*state-p* σ) (*x* ∈ *state-*χ σ))›
  assumes ‹*S* ⊆ *run-state-set* ϱ›
  shows ‹*measure-pmf.expectation* (*run-state-pmf* ϱ) (*aux S*) ≤ φ *1 True* ^ *card S*›
  ⟨*proof*⟩

Lemma 2:

lemma *run-steps-preserves-expectation-le'* :
  fixes *q* :: *real* **and** *h* :: ‹*real* ⇒ *real*›
  assumes *h*:
    ‹*0* < *q*› ‹*q* ≤ *1*›
    ‹*concave-on* {*0* .. *1* / *q*} *h*›
    ‹$\bigwedge$ *x*. ⟦*0* ≤ *x*; *x* ∗ *q* ≤ *1*⟧ ⟹ *h x* ≥ *0*›
  defines
    ‹*aux* ≡ λ*S* σ. ($\prod$ *x* ∈ *S*. *of-bool* (*state-p* σ ≥ *q*) ∗ *h* (*of-bool* (*x* ∈ *state-*χ σ) / *state-p*
σ))›

  assumes ‹*S* ⊆ *run-state-set* ϱ›
  shows ‹($\int$ τ. *aux S* τ ∂*run-state-pmf* ϱ) ≤ (*h 1*) ^ *card S*› (**is** ‹*?L* ≤ *?R*›)
⟨*proof*⟩

Analysis of the case where $n \leq card\ (set\ xs)$:

context
  fixes *xs* :: ‹*'a list*›
begin

private abbreviation ‹*A* ≡ *real* (*card* (*set xs*))›

context
  assumes *set-larger-than-n*: ‹*card* (*set xs*) ≥ *n*›
begin

private definition ‹*q* = *real n* / (*4* ∗ *card* (*set xs*))›

lemma *q-range*: ‹*q* ∈ {*0<..1/4*}›
  ⟨*proof*⟩

lemma *mono-nonnegI*:
  assumes ‹$\bigwedge$*x*. *x* ∈ *I* ⟹ *h' x* ≥ *0*›
  assumes ‹$\bigwedge$*x*. *x* ∈ *I* ⟹ (*h has-real-derivative* (*h' x*)) (*at x*)›
  assumes ‹*x* ∈ *I* ∩ {*0..*}› ‹*convex I*› ‹*0* ∈ *I*› ‹*h 0* ≥ *0*›
  shows ‹*h x* ≥ *0*›
⟨*proof*⟩

lemma *upper-tail-bound-helper*:
  assumes ‹$x \in \{0{<}..1::real\}$›
  defines ‹$h \equiv (\lambda x. - q * x^2 / 3 - ln\ (1 + q * x) + q * ln\ (1 + x) * (1 + x))$›
  shows ‹$h\ x \geq 0$›
⟨*proof*⟩ **definition** ϑ **where** ‹$\vartheta\ t\ x = 1 + q * x * (exp\ (t / q) - 1)$›

lemma *ϑ-concave*: ‹*concave-on* $\{0..1 / q\}$ $(\vartheta\ t)$›
  ⟨*proof*⟩

lemma *ϑ-ge-exp-1*:
  assumes ‹$x \in \{0..1/q\}$›
  shows ‹$exp\ (t * x) \leq \vartheta\ t\ x$›
⟨*proof*⟩

lemma *ϑ-ge-exp*:
  assumes ‹$y \geq q$›
  shows ‹$exp\ (t / y) \leq \vartheta\ t\ (1 / y)$›
  ⟨*proof*⟩

lemma *ϑ-nonneg*:
  assumes ‹$x \in \{0..1/q\}$›
  shows ‹$\vartheta\ t\ x \geq 0$› ‹$\vartheta\ t\ x > 0$›
⟨*proof*⟩

lemma *ϑ-0*: ‹$\vartheta\ t\ 0 = 1$› ⟨*proof*⟩

lemma *tail-bound-aux*:
  assumes ‹*run-state-set* $\varrho \subseteq$ *set xs*› ‹$c > 0$›
  defines ‹$A' \equiv$ *real* (*card* (*run-state-set* ϱ))›
  shows ‹*measure* (*run-state-pmf* ϱ) $\{\omega.\ exp\ (t * estimate\ \omega) \geq c \wedge state\text{-}p\ \omega \geq q\} \leq \vartheta\ t$
1 powr A'/c›
    (**is** ‹$?L \leq ?R$›)
⟨*proof*⟩

Lemma 3:

lemma *upper-tail-bound*:
  assumes ‹$\varepsilon \in \{0{<}..1::real\}$›
  assumes ‹*run-state-set* $\varrho \subseteq$ *set xs*›
  shows ‹*measure* (*run-state-pmf* ϱ) $\{\omega.\ estimate\ \omega \geq (1+\varepsilon)*A \wedge state\text{-}p\ \omega \geq q\} \leq$
$exp(-real\ n/12*\varepsilon^2)$›
    (**is** ‹$?L \leq ?R$›)
⟨*proof*⟩

Lemma 4:

lemma *low-p*:
  shows ‹*measure* (*run-steps xs*) $\{\sigma.\ state\text{-}p\ \sigma < q\} \leq real\ (length\ xs) * exp(-real\ n/12)$›
    (**is** ‹$?L \leq ?R$›)
⟨*proof*⟩

lemma *lower-tail-bound-helper*:
  assumes ‹$x \in \{0{<}..{<}1::real\}$›
  defines ‹$h \equiv (\lambda x. - q * x^2 / 2 - ln\ (1 - q * x) + q * ln\ (1 - x) * (1 - x))$›
  shows ‹$h\ x \geq 0$›

⟨*proof*⟩

Lemma 5:

lemma *lower-tail-bound*:
assumes ‹$\varepsilon \in \{0<..<1::real\}$›
shows ‹*measure* (*run-steps xs*) $\{\omega.$ *estimate* $\omega \leq (1-\varepsilon) * A \wedge$ *state-p* $\omega \geq q\} \leq exp(-real$ $n/8*\varepsilon^2)$›
(**is** ‹?L ≤ ?R›)
⟨*proof*⟩

lemma *correctness-aux*:
assumes ‹$\varepsilon \in \{0<..<1::real\}$› ‹$\delta \in \{0<..<1::real\}$›
assumes ‹*real* $n \geq 12/\varepsilon\hat{\ }2 * ln$ (3∗*real* (*length xs*) /δ)›
shows ‹*measure* (*run-steps xs*) {ω. |*estimate* ω − *A*| > ε∗*A* } ≤ δ›
(**is** ‹?L ≤ ?R›)
⟨*proof*⟩

end

lemma *deterministic-phase*:
assumes ‹*card* (*run-state-set* σ) < *n*›
shows ‹*run-state-pmf* σ = *return-pmf* (*State* (*run-state-set* σ) *1*)›
⟨*proof*⟩

Theorem 1:

theorem *correctness*:
fixes ε δ :: *real*
assumes ‹$\varepsilon \in \{0<..<1\}$› ‹$\delta \in \{0<..<1\}$›
assumes ‹*real* $n \geq 12 / \varepsilon^2 * ln$ (3 ∗ *real* (*length xs*) / δ)›
shows ‹*measure* (*run-steps xs*) {ω. |*estimate* ω − *A*| > ε ∗ *A*} ≤ δ›
⟨*proof*⟩

lemma *p-pos*: ‹∃ *M*∈{0<..1}. *AE* ω *in run-steps xs*. *state-p* ω ≥ *M*›
⟨*proof*⟩

lemma *run-steps-expectation-sing*:
assumes *i*: ‹*i* ∈ *set xs*›
shows ‹*measure-pmf.expectation* (*run-steps xs*) (λω. *of-bool* (*i* ∈ *state-χ* ω) / *state-p* ω) = *1*›
(**is** ‹?L = -›)
⟨*proof*⟩

Subsection A.3:

corollary *unbiasedness*:
fixes σ :: ‹'*a run-state*›
shows ‹*measure-pmf.expectation* (*run-steps xs*) *estimate* = *real* (*card* (*set xs*))›
(**is** ‹?L = -›)
⟨*proof*⟩

end

end

end

3 The Original CVM Algorithm

In this section, we verify the algorithm as presented by Chakrabory et al. [1] (replicated, here, in Algorithm 2), with the following caveat:

In the original algorithm the elements are removed with probability $f := \frac{1}{2}$ in the subsampling step. The version verified here allows for any $f \in [\frac{1}{2}, e^{-1/12}]$.

Algorithm 2 Original CVM algorithm.

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

1: $\chi \leftarrow \{\}, p \leftarrow 1, n \geq \left\lceil \frac{12}{\varepsilon^2} \ln(\frac{6l}{\delta}) \right\rceil$
2: **for** $i \leftarrow 1$ to l **do**
3:     $b \xleftarrow{\$} \mathrm{Ber}(p)$ ▷ insert a_i with probability p (and remove it otherwise)
4:     **if** b **then**
5:         $\chi \leftarrow \chi \cup \{a_i\}$
6:     **else**
7:         $\chi \leftarrow \chi - \{a_i\}$
8:     **if** $|\chi| = n$ **then**
9:         $\chi \xleftarrow{\$} \mathrm{subsample}(\chi)$ ▷ keep each element of χ indep. with prob. f
10:         $p \leftarrow pf$
11:     **if** $|\chi| = n$ **then**
12:         **return** $\perp$
13: **return** $\frac{|\chi|}{p}$ ▷ estimate cardinality of A

The first step of the proof is identical to the original proof [1], where the above algorithm is approximated by a second algorithm, where lines 11–12 are removed, i.e., the two algorithms behave identically, unless the very improbable event—where the subsampling step fails to remove any elements—occurs. It is possible to show that the total variational distance between the two algorithms is at most $\frac{\delta}{2}$.

In the second step, we verify that the probability that the second algorithm returns an estimate outside of the desired interval is also at most $\frac{\delta}{2}$. This, of course, works by noticing that it is an instance of the abstract algorithm we introduced in Section 2. In combination, we conclude a failure probability of δ for the unmodified version of the algorithm.

On the other hand, the fact that the number of elements in the buffer is at most n can be seen directly from Algorithm 2.

theory *CVM-Original-Algorithm*
  imports *CVM-Abstract-Algorithm*
begin

context
  fixes f :: *real*
  fixes n :: *nat*
  assumes *f-range*: ‹$f \in \{1/2..exp(-1/12)\}$›
  assumes *n-gt-0*: ‹$n > 0$›
begin

Line 1:

```
definition initial-state :: ‹'a state› where
  ‹initial-state = State {} 1›
```

Lines 3–7:

```
fun step-1 :: ‹'a ⇒ 'a state ⇒ 'a state spmf› where
  ‹step-1 a (State χ p) =
    do {
      b ← bernoulli-pmf p;
      let χ = (if b then χ ∪ {a} else χ − {a});

      return-spmf (State χ p)
    }›

definition subsample :: ‹'a set ⇒ 'a set spmf› where
  ‹subsample χ =
    do {
      keep-in-χ ← prod-pmf χ (λ-. bernoulli-pmf f);
      return-spmf (Set.filter keep-in-χ χ)
    }›
```

Lines 8–10:

```
fun step-2 :: ‹'a state ⇒ 'a state spmf› where
  ‹step-2 (State χ p) =
    do {
      if card χ = n then do {
        χ ← subsample χ;
        return-spmf (State χ (p ∗ f))
      } else
        return-spmf (State χ p)
    }›
```

Lines 11–12:

```
fun step-3 :: ‹'a state ⇒ 'a state spmf› where
  ‹step-3 (State χ p) =
    do {
      if card χ = n
      then fail-spmf
      else return-spmf (State χ p)
    }›
```

Lines 1–12:

```
definition run-steps :: ‹'a list ⇒ 'a state spmf› where
  ‹run-steps xs ≡ foldM-spmf (λx σ. step-1 x σ ≫= step-2 ≫= step-3) xs initial-state›
```

Line 13:

```
definition estimate :: ‹'a state ⇒ real› where
  ‹estimate σ = card (state-χ σ) / state-p σ›

definition run-algo :: ‹'a list ⇒ real spmf› where
  ‹run-algo xs = map-spmf estimate (run-steps xs)›
```

schematic-goal *step-1-m-def*: ‹*step-1 x σ = ?x*›
⟨*proof*⟩

schematic-goal *step-2-m-def*: ‹*step-2 σ = ?x*›
⟨*proof*⟩

schematic-goal *step-3-m-def*: ‹*step-3 σ = ?x*›
⟨*proof*⟩

lemma *ord-spmf-remove-step3*:
‹*ord-spmf* (=) (*step-1 x σ* ⋙= *step-2* ⋙= *step-3*) (*step-1 x σ* ⋙= *step-2*)›
⟨*proof*⟩

lemma *ord-spmf-run-steps*:
‹*ord-spmf* (=) (*run-steps xs*) (*foldM-spmf* (*λx σ. step-1 x σ* ⋙= *step-2*) *xs initial-state*)›
⟨*proof*⟩

lemma *f-range-simple*: ‹$f \in \{1/2..<1\}$›
⟨*proof*⟩

Main result:

theorem *correctness*:
fixes *xs* :: ‹*'a list*›
assumes ‹$\varepsilon \in \{0<..<1\}$› ‹$\delta \in \{0<..<1\}$›
assumes ‹*real* $n \geq 12 / \varepsilon^2 * ln$ (6 ∗ *real* (*length xs*) / δ)›
defines ‹$A \equiv$ *real* (*card* (*set xs*))›
shows ‹$\mathcal{P}$(ω *in run-algo xs. fails-or-satisfies* (λR. $|R - A| > \varepsilon * A$) ω) ≤ δ›
(**is** ‹*?L* ≤ *?R*›)
⟨*proof*⟩

lemma *space-usage*:
‹*AE σ in measure-spmf* (*run-steps xs*). *card* (*state-χ σ*) < *n* ∧ *finite* (*state-χ σ*)›
⟨*proof*⟩

end

end

4 The New Unbiased Algorithm

In this section, we introduce the new algorithm variant promised in the abstract.

The main change is to replace the subsampling step of the original algorithm, which removes each element of the buffer independently with probability f. Instead, we choose a random nf-subset of the buffer (see Algorithm 3). (This means f, n must be chosen, such that nf is an integer.)

Algorithm 3 New CVM algorithm.

Input: Stream elements $a_1, \ldots, a_l$, $0 < \varepsilon$, $0 < \delta < 1$, f subsampling param.
Output: An estimate R, s.t., $\mathcal{P}(|R - |A|| > \varepsilon|A|) \le \delta$ where $A := \{a_1, \ldots, a_l\}$.

$\chi \leftarrow \{\}, p \leftarrow 1, n \geq \left\lceil \frac{12}{\varepsilon^2} \ln(\frac{3l}{\delta}) \right\rceil$
for $i \leftarrow 1$ to l **do**
  $b \xleftarrow{\$} \text{Ber}(p)$ ▷ 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**
    $\chi \xleftarrow{\$} \text{subsample}(\chi)$ ▷ Choose a random nf-subset of χ
    $p \leftarrow pf$
return $\frac{|\chi|}{p}$ ▷ estimate cardinality of A

The fact that this still preserves the required inequality for the subsampling operation (Eq. 1) follows from the negative associativity of permutation distributions [2, Th. 10].

(See also our formalization of the concept [3].)

Because the subsampling step always removes elements unconditionally, the second check, whether the subsampling succeeded of the original algorithm is not necessary anymore.

This improves the space usage of the algorithm, because the first reduction argument from Section 3 is now obsolete. Moreover the resulting algorithm is now unbiased, because it is an instance of the abstract algorithm of Section 2.

theory *CVM-New-Unbiased-Algorithm*
  imports
    CVM-Abstract-Algorithm
    Probabilistic-Prime-Tests.Generalized-Primality-Test
    Negative-Association.Negative-Association-Permutation-Distributions
begin

unbundle *no vec-syntax*

context
  fixes f :: *real* **and** n :: *nat*
  assumes *f-range*: ‹$f \in \{1/2..<1\}$› ‹$n * f \in \mathbb{N}$› **and** *n-gt-0*: ‹$n > 0$›
begin

definition ‹*initial-state* = *State* {} *1*› — Setup initial state $\chi = \emptyset$ 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*)›

definition ‹*subsample-size* = (*THE x*. *real x* = *n* ∗ *f*)›

declare *subsample.simps* [*simp del*]

lemma *subsample-size-eq*:
‹*real subsample-size* = *n* ∗ *f*›
⟨*proof*⟩

lemma *subsample-size*:
‹*subsample-size* < *n*› ‹*2* ∗ *subsample-size* ≥ *n*›
⟨*proof*⟩

lemma *subsample-finite-nonempty*:
assumes ‹*card U* = *n*›
shows
‹{*T*. *T* ⊆ *U* ∧ *card T* = *subsample-size*} ≠ {}› (**is** ‹*?C* ≠ {}›)
‹*finite* {*T*. *T* ⊆ *U* ∧ *card T* = *subsample-size*}›
‹*subsample U* = *pmf-of-set* {*T*. *T* ⊆ *U* ∧ *card T* = *subsample-size*}›
‹*finite* (*set-pmf* (*subsample U*))›
⟨*proof*⟩

lemma *int-prod-subsample-eq-prod-int*:
fixes *g* :: ‹*bool* ⇒ *real*›
assumes ‹*card U* = *n*› ‹*S* ⊆ *U*› ‹*range g* ⊆ {*0*..}›
shows ‹($\int \omega$. ($\prod s{\in}S$. $g(s \in \omega)$) ∂*subsample U*) ≤ ($\prod s{\in}S$. ($\int \omega$. $g\ \omega$ ∂*bernoulli-pmf f*))›
(**is** ‹*?L* ≤ *?R*›)
⟨*proof*⟩

schematic-goal *step-n-def*: ‹*step x σ = ?x*›
  ⟨*proof*⟩

interpretation *abs*: *cvm-algo-abstract n f subsample*
  rewrites ‹*abs.run-steps = run-steps*› **and** ‹*abs.estimate = estimate*›
⟨*proof*⟩

theorem *unbiasedness*: ‹*measure-pmf.expectation* (*run-algo xs*) *id* = *card* (*set xs*)›
  ⟨*proof*⟩

theorem *correctness*:
  assumes ‹$\varepsilon \in \{0{<}..{<}1{::}real\}$› ‹$\delta \in \{0{<}..{<}1{::}real\}$›
  assumes ‹$real\ n \geq 12\ /\ \varepsilon^2 * ln\ (3 * real\ (length\ xs)\ /\ \delta)$›
  defines ‹$A \equiv real\ (card\ (set\ xs))$›
  shows ‹$\mathcal{P}(R\ in\ run\text{-}algo\ xs.\ |R - A| > \varepsilon * A) \leq \delta$›
  ⟨*proof*⟩

lemma *space-usage*:
  ‹$AE\ \sigma\ in\ run\text{-}steps\ xs.\ card\ (state\text{-}\chi\ \sigma) < n \wedge finite\ (state\text{-}\chi\ \sigma)$›
⟨*proof*⟩

end

end

References

[1] S. Chakraborty, N. V. Vinodchandran, and K. S. Meel. Distinct elements in streams: An algorithm for the (text) book. In S. Chechik, G. Navarro, E. Rotenberg, and G. Herman, editors, *ESA*, volume 244 of *LIPIcs*, pages 34:1–34:6. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[2] D. P. Dubhashi, V. Priebe, and D. Ranjan. Negative dependence through the fkg inequality. *BRICS Report Series*, 3, 1996.

[3] E. Karayel. Negatively associated random variables. *Archive of Formal Proofs*, January 2025. https://isa-afp.org/entries/Negative_Association.html, Formal proof development.

A Informal Proof

This section includes an informal version of the proof for the tail bounds and unbiasedness of the abstract algorithm (Algorithm 1) for interested readers.

This means we assume the subsample(χ) operation fulfills Eq. 1 and always returns a subset of χ.

Notation: For a finite set S, the probability space of uniformly sampling from the set is denoted by $U(S)$, i.e., for each $s \in S$ we have $\mathcal{P}_{U(S)}(s) = |S|^{-1}$. We write $\mathrm{Ber}(p)$ for the Bernoulli probability space, over the set $\{0, 1\}$, i.e., $P_{\mathrm{Ber}(p)}(\{1\}) = p$. $I(P)$ is the indicator function for a predicate P, i.e., $I(\mathrm{true}) = 1$ and $I(\mathrm{false}) = 0$.

Like in the formalization, we will denote the first five lines of the loop (3–7) as step 1, the last four lines (8–10) as step 2. For the distribution of the state of the algorithm after processing i elements of the sequence, we will write Ω_i. The elements of the probability spaces are pairs composed of a set and the number of subsampling steps, representing χ and p respectively.

For example: $\Omega_0 = U(\{(\emptyset, 1)\})$ is the initial state, $\Omega_1 = U(\{(\{a_1\}, 1)\})$, etc., and Ω_l denotes the final state. We introduce χ and p as random variables defined over such probability spaces Ω, in particular, χ (resp. p) is the projection to the first (resp. second) component.

The state of the algorithm after processing only step 1 of the i-th loop iteration is denoted by Ω'_i. So the sequence of states is represented by the distributions $\Omega_0, \Omega'_1, \Omega_1, \ldots, \Omega'_l, \Omega_l$.

A.1 Loop Invariant

After these preliminaries, we can go to the main proof, whose core is a probabilistic loop invariant for Algorithm 1 that can be verified inductively.

Lemma 1. Let $\varphi : (0, 1] \times \{0, 1\} \to \mathbb{R}_{\geq 0}$ be a function, fulfilling the following conditions:

1. $(1 - \alpha)\varphi(x, 0) + \alpha\varphi(x, 1) \leq \varphi(x/\alpha, 1)$ for all $0 < \alpha < 1$, $0 < x \leq 1$, and
2. $\varphi(x, 0) \leq \varphi(y, 0)$ for all $0 < x < y \leq 1$.

Then for all $k \in \{0, \ldots, l\}$, $S \subseteq \{a_1, .., a_k\}$, $\Omega \in \{\Omega_k, \Omega'_k\}$:

$$\mathbb{E}_{\Omega}\left[\prod_{s \in S} \varphi(p, I(s \in \chi))\right] \leq \varphi(1, 1)^{|S|}$$

Proof. We show the result using induction over k. Note that we show the statement for arbitrary S, i.e., the induction statements are:

$$P(k) \quad :\leftrightarrow \quad \left(\forall S \subseteq \{a_1, .., a_k\}.\ \mathbb{E}_{\Omega_k}\left[\prod_{s \in S} \varphi(p, I(s \in \chi))\right] \leq \varphi(1, 1)^{|S|}\right)$$

$$Q(k) \quad :\leftrightarrow \quad \left(\forall S \subseteq \{a_1, .., a_k\}.\ \mathbb{E}_{\Omega'_k}\left[\prod_{s \in S} \varphi(p, I(s \in \chi))\right] \leq \varphi(1, 1)^{|S|}\right)$$

and we will show $P(0), Q(1), P(1), Q(2), P(2), \dots, Q(l), P(l)$ successively.

Induction start $P(0)$:
We have $S \subseteq \emptyset$, and hence

$$\mathbb{E}_{\Omega_0}\left[\prod_{s\in S}\varphi(p, I(s\in\chi))\right] = \mathbb{E}_{\Omega_0}[1] = 1 \leq \varphi(1,1)^0.$$

Induction step $P(k) \to Q(k+1)$:
Let $S \subseteq \{a_1, \dots, a_{k+1}\}$ and define $S' := S - \{a_{k+1}\}$. Note that Ω'_{k+1} can be constructed from Ω_k as a compound distribution, where a_{k+1} is included in the buffer, with the probability p, which is itself a random variable over the space Ω_k.
In particular, for example:

$$\mathcal{P}_{\Omega'_{k+1}}(P(\chi,p)) = \int_{\Omega_k}\int_{\mathrm{Ber}(p(\omega))} P(\chi(\omega) - \{a_{k+1}\} \cup f(\tau), p(\omega))\, d\tau\, d\omega$$

for all predicates P where we define $f(1) = \{a_{k+1}\}$ and $f(0) = \emptyset$.
We distinguish the two cases $a_{k+1} \in S$ and $a_{k+1} \notin S$. If $a_{k+1} \in S$:

$$\begin{aligned}
& \mathbb{E}_{\Omega'_{k+1}}\left[\prod_{s\in S}\varphi(p, I(s\in\chi))\right] \\
= \quad & \int_{\Omega_k}\left(\prod_{s\in S'}\varphi(p, I(s\in\chi))\right)\int_{\mathrm{Ber}(p(\omega))}\varphi(p,\tau)\, d\tau\, d\omega \\
= \quad & \int_{\Omega_k}\left(\prod_{s\in S'}\varphi(p, I(s\in\chi))\right)((1-p)\varphi(p,0) + p\varphi(p,1))\, d\omega \\
\underset{\text{Cond 1}}{\leq} \quad & \int_{\Omega_k}\left(\prod_{s\in S'}\varphi(p, I(s\in\chi))\right)\varphi(1,1)\, d\omega \\
\underset{\text{IH}}{\leq} \quad & \varphi(1,1)^{|S'|}\varphi(1,1) = \varphi(1,1)^{|S|}
\end{aligned}$$

If $a_{k+1} \notin S$ then $S' = S$ and:

$$\mathbb{E}_{\Omega'_{k+1}}\left[\prod_{s\in S}\varphi(p, I(s\in\chi))\right] = \int_{\Omega_k}\prod_{s\in S}\varphi(p, I(s\in\chi))\, d\omega \leq_{\text{IH}} \varphi(1,1)^{|S'|} = \varphi(1,1)^{|S|}$$

Induction step $Q(k+1) \to P(k+1)$:
Let $S \subseteq \{a_1, \dots, a_{k+1}\}$.
Let us again note that Ω_{k+1} is a compound distribution over Ω'_{k+1}. In general, for all predicates P:

$$\mathcal{P}_{\Omega_{k+1}}(P(\chi,p)) =$$
$$\int_{\Omega'_{k+1}} I(|\chi(\omega)| < n) P(\chi(\omega), p(\omega)) + I(|\chi(\omega)| = n)\int_{\mathrm{subsample}(\chi(\omega))} P(\tau, fp(\omega))\, d\tau\, d\omega.$$

With this we can can now verify the induction step:

$$
\begin{array}{rll}
& \mathbb{E}_{\Omega_{k+1}}\left[\prod_{s\in S}\varphi(p, I(s\in\chi))\right] & \\
= & \int_{\Omega'_{k+1}} I(|\chi|<n)\prod_{s\in S}\varphi(p, I(s\in\chi))\,d\omega & \\
+ & \int_{\Omega'_{k+1}} I(|\chi|=n)\prod_{s\in S\setminus\chi(\omega)}\varphi(pf,0)\int_{\text{subsample}(\chi)}\prod_{s\in S\cap\chi}\varphi(pf, I(s\in\tau))d\tau\,d\omega & \\
\leq & \int_{\Omega'_{k+1}} I(|\chi|<n)\prod_{s\in S}\varphi(p, I(s\in\chi))\,d\omega & \text{Eq. 1} \\
+ & \int_{\Omega'_{k+1}} I(|\chi|=n)\prod_{s\in S\setminus\chi(\omega)}\varphi(pf,0)\prod_{s\in S\cap\chi}\int_{\text{Ber(f)}}\varphi(pf,\tau)d\tau\,d\omega & \\
\leq & \int_{\Omega'_{k+1}} I(|\chi|<n)\prod_{s\in S}\varphi(p, I(s\in\chi))\,d\omega & \text{Cond 2} \\
+ & \int_{\Omega'_{k+1}} I(|\chi|=n)\prod_{s\in S\setminus\chi(\omega)}\varphi(p,0)\prod_{s\in S\cap\chi}((1-f)\varphi(pf,0)+f\varphi(pf,1))\,d\omega & \\
\leq & \int_{\Omega'_{k+1}} I(|\chi|<n)\prod_{s\in S}\varphi(p, I(s\in\chi))\,d\omega & \text{Cond 1} \\
+ & \int_{\Omega'_{k+1}} I(|\chi|=n)\prod_{s\in S\setminus\chi(\omega)}\varphi(p,0)\prod_{s\in S\cap\chi}\varphi(p,1)\,d\omega & \\
= & \int_{\Omega'_{k+1}} I(|\chi|<n)\prod_{s\in S}\varphi(p, I(s\in\chi))\,d\omega & \\
+ & \int_{\Omega'_{k+1}} I(|\chi|=n)\prod_{s\in S}\varphi(p, I(s\in\chi))\,d\omega & \\
= & \mathbb{E}_{\Omega'_{k+1}}\left[\prod_{s\in S}\varphi(p, I(s\in\chi))\right]\leq\varphi(1,1)^{|S|} & \text{IH}
\end{array}
$$

□

A corollary and more practical version of the previous lemma is:

Lemma 2. Let $q \leq 1$ and $h : [0, q^{-1}] \to \mathbb{R}_{\geq 0}$ be concave then for all $k \in \{0, \ldots, l\}$, $S \subseteq \{a_1, .., a_k\}$, $\Omega \in \{\Omega_k, \Omega'_k\}$:

$$\mathbb{E}_{\Omega}\left[\prod_{s\in S} I(p>q)h(p^{-1}I(s\in\chi))\right]\leq h(1)^{|S|}$$

Proof. Follows from Lemma 1 for $\varphi(p,\tau) := I(p > q)h(\tau p^{-1})$. We just need to check Conditions 1 and 2. Indeed,

$$
\begin{array}{rcl}
(1-\alpha)\varphi(x,0)+\alpha\varphi(x,1) & = & (1-\alpha)I(x>q)h(0)+\alpha I(x>q)h(x^{-1}) \\
& \leq & I(x>q)h(\alpha x^{-1})\leq I(x>q\alpha)h(\alpha x^{-1})=\varphi(x/\alpha,1)
\end{array}
$$

for $0 < \alpha < 1$ and $0 < x \leq 1$, where we used that $x > q$ implies $x > q\alpha$; and

$$\varphi(x,0)=I(x>q)h(0)\leq I(y>q)h(0)=\varphi(y,0)$$

for $0 < x < y \leq 1$, where we used that $x > q$ implies $y > q$. □

It should be noted that this is a probabilistic recurrence relation, but the main innovation is that we establish a relation, with respect to general classes of functions of the state variables.

A.2 Concentration

Let us now see how we can obtain concentration bounds using Lemma 2, i.e., that the result of the algorithm is concentrated around the cardinality of $A = \{a_1, \ldots, a_l\}$. This will be done using the Cramér–Chernoff method for the probability that the estimate is above $(1+\varepsilon)|A|$ (resp. below $(1-\varepsilon)|A|$) assuming p is not too small and a tail estimate for p being too small.

It should be noted that concentration is trivial, if $|A| < n$, i.e., if we never need to do sub-sampling, so we assume $|A| \geq n$.

Define $q := n/(4|A|)$ and notice that $q \leq \frac{1}{4}$.

Let us start with the upper tail bound:

Lemma 3. For any $\Omega \in \{\Omega_0, \ldots, \Omega_l\} \cup \{\Omega'_1, \ldots, \Omega'_l\}$ and $0 < \varepsilon \leq 1$:

$$L := \mathcal{P}_\Omega\left(p^{-1}|\chi| \geq (1+\varepsilon)|A| \wedge p \geq q\right) \leq \exp\left(-\frac{n}{12}\varepsilon^2\right)$$

Proof. By assumption there exists a k such that $\Omega \in \{\Omega_k, \Omega'_k\}$. Let $A' = A \cap \{a_1, \ldots, a_k\}$. Moreover, we define:

$$t := q\ln(1+\varepsilon)$$
$$h(x) := 1 + qx(e^{t/q} - 1)$$

To get a tail estimate, we use the Cramér–Chernoff method:

$$\begin{aligned}
L &\underset{t>0}{\leq} \mathcal{P}_\Omega\left(\exp(tp^{-1}|\chi|) \geq \exp(t(1+\varepsilon)|A|) \wedge p \geq q\right) \\
&\leq \mathcal{P}_\Omega\left(I(p \geq q)\exp(tp^{-1}|\chi|) \geq \exp(t(1+\varepsilon)|A|)\right) \\
&\underset{\text{Markov}}{\leq} \exp(-t(1+\varepsilon)|A|)\,\mathbb{E}_\Omega\left[I(p \geq q)\exp(tp^{-1}|\chi|)\right] \\
&\leq \exp(-t(1+\varepsilon)|A|)\,\mathbb{E}_\Omega\left[\prod_{s\in A'} I(p \geq q)\exp(tp^{-1}I(s \in \chi))\right] \\
&\leq \exp(-t(1+\varepsilon)|A|)\,\mathbb{E}_\Omega\left[\prod_{s\in A'} I(p \geq q)h(p^{-1}I(s \in \chi))\right] \\
&\underset{\text{Le. 2}}{\leq} \exp(-t(1+\varepsilon)|A|)h(1)^{|A'|} \\
&\underset{h(1)\geq 1}{\leq} (\exp(\ln(h(1)) - t(1+\varepsilon)))^{|A|}
\end{aligned}$$

So we just need to show that (using $|A| = \frac{n}{4q}$):

$$\ln(h(1)) - t(1+\varepsilon) \leq \frac{-q\varepsilon^2}{3}$$

The latter can be established by analyzing the function

$$f(\varepsilon) := -\ln(1+q\varepsilon) + q\ln(1+\varepsilon)(1+\varepsilon) - \frac{q\varepsilon^2}{3} = -\ln(h(1)) + t(1+\varepsilon) - \frac{q\varepsilon^2}{3}.$$

For which it is easy to check $f(0) = 0$ and the derivative with respect to ε is non-negative in the range $0 \leq q \leq 1/4$ and $0 < \varepsilon \leq 1$, i.e., $f(\varepsilon) \geq 0$. $\square$

Using the previous result we can also estimate bounds for p becoming too small:

Lemma 4.

$$\mathcal{P}_{\Omega_l}(p < q) \leq l \exp\left(-\frac{n}{12}\right)$$

Proof. We will use a similar strategy as in the Bad_2 bound from the original CVM paper [1]. Let j be maximal, s.t., $q \leq f^j$. Hence $f^{j+1} < q$ and:

$$f^j \leq 2ff^j < 2q = \frac{n}{2|A|}. \tag{2}$$

First, we bound the probability of jumping from $p = f^j$ to $p = f^{j+1}$ at a specific point in the algorithm, e.g., while processing k stream elements. It can only happen if $|\chi| = n$, $p = f^j$ in Ω'_k. Then

$$\begin{aligned}
\mathcal{P}_{\Omega'_k}(|\chi| \geq n \wedge p = f^j) &\leq \mathcal{P}(p^{-1}|\chi| \geq f^{-j}n \wedge p \geq q) \\
&\underset{\text{Eq. 2}}{\leq} \mathcal{P}(p^{-1}|\chi| \geq 2|A| \wedge p \geq q) \\
&\underset{\text{Le. 3}}{\leq} \exp(-n/12)
\end{aligned}$$

The probability that this happens ever in the entire process is then at most l times the above which proves the lemma. $\square$

Lemma 5. Let $0 < \varepsilon < 1$ then:

$$L := \mathcal{P}_{\Omega_l}(p^{-1}|\chi| \leq (1-\varepsilon)|A| \wedge p \geq q) \leq \exp\left(-\frac{n}{8}\varepsilon^2\right)$$

Proof. Let us define

$$\begin{aligned}
t &:= q \ln(1-\varepsilon) < 0 \\
h(x) &:= 1 + qx(e^{t/q} - 1)
\end{aligned}$$

Note that $h(x) \geq 0$ for $0 \leq x \leq q^{-1}$ (can be checked by verifying it is true for $h(0)$ and $h(q^{-1})$ and the fact that the function is affine.)

With these definitions we again follow the Cramér–Chernoff method:

$$\begin{aligned}
L &\underset{t<0}{=} \mathcal{P}_{\Omega_l}\left(\exp(tp^{-1}|\chi|) \geq \exp(t(1-\varepsilon)|A|) \wedge p \geq q\right) \\
&\leq \mathcal{P}_{\Omega_l}\left(I(p \geq q)\exp(tp^{-1}|\chi|) \geq \exp(t(1-\varepsilon)|A|) \wedge p > q\right) \\
&\underset{\text{Markov}}{\leq} \exp(-t(1-\varepsilon)|A|)\,\mathbb{E}_\Omega\left[I(p \geq q)\exp(tp^{-1}|\chi|)\right] \\
&= \exp(-t(1-\varepsilon)|A|)\,\mathbb{E}_\Omega\left[\prod_{s\in A} I(p \geq q)\exp(tp^{-1}I(s \in \chi))\right] \\
&\leq \exp(-t(1-\varepsilon)|A|)\,\mathbb{E}_\Omega\left[\prod_{s\in A} I(p \geq q)h(p^{-1}I(s \in \chi))\right] \\
&\underset{\text{Le. 2}}{\leq} \exp(-t(1-\varepsilon)|A|)(h(1))^{|A|} \\
&= \exp(\ln(h(1)) - t(1-\varepsilon))^{|A|}
\end{aligned}$$

Substituting t and h and using $|A| = \frac{n}{4q}$, we can see that the lemma is true if

$$f(\varepsilon) := q\ln(1-\varepsilon)(1-\varepsilon) - \ln(1-q\varepsilon) - \frac{q}{2}\varepsilon^2 = t(1-\varepsilon) - \ln(h(1)) - \frac{q}{2}\varepsilon^2$$

is non-negative for $0 \leq q \leq \frac{1}{4}$ and $0 < \varepsilon < 1$. This can be verified by checking that $f(0) = 0$ and that the derivative with respect to ε is non-negative. □

We can now establish the concentration result:

Theorem 1. *Let $0 < \varepsilon < 1$ and $0 < \delta < 1$ and $n \geq \frac{12}{\varepsilon^2}\ln\left(\frac{3l}{\delta}\right)$ then:*

$$L = \mathcal{P}_{\Omega_l}\left(|p^{-1}|\chi| - |A|| \geq \varepsilon|A|\right) \leq \delta$$

Proof. Note that the theorem is trivial if $|A| < n$. If not:

$$\begin{array}{lll} L & \leq & \mathcal{P}_{\Omega_l}\left(|p^{-1}|\chi| \leq (1-\varepsilon)|A| \wedge p \geq q\right) \\ & + & \mathcal{P}_{\Omega_l}\left(|p^{-1}|\chi| \geq (1+\varepsilon)|A| \wedge p \geq q\right) + \mathcal{P}_{\Omega_l}\left(p < q\right) \\ & \underset{Le.\ 3\text{–}5}{\leq} & \exp\left(-\frac{n}{8}\varepsilon^2\right) + \exp\left(-\frac{n}{12}\varepsilon^2\right) + l\exp\left(-\frac{n}{12}\right) \\ & \leq & \frac{\delta}{3} + \frac{\delta}{3} + \frac{\delta}{3} \end{array}$$

□

A.3 Unbiasedness

Let M be large enough such that $p^{-1} \leq M$ a.s. (e.g., we can choose $M = f^{-l}$). Then we can derive from Lemma 2 using $h(x) = x$ and $h(x) = M + 1 - x$ that for all $s \in A$:

$$\begin{array}{rcl} \mathbb{E}_{\Omega_l}[p^{-1}I(s \in \chi)] & = & \mathbb{E}_{\Omega_l}[I(p \geq M^{-1})p^{-1}I(s \in \chi)] \leq 1 \\ \mathbb{E}_{\Omega_l}[M + 1 - p^{-1}I(s \in \chi)] & = & \mathbb{E}_{\Omega_l}[I(p \geq M^{-1})(M + 1 - p^{-1}I(s \in \chi))] \leq M \end{array}$$

which implies $\mathbb{E}_{\Omega_l}[p^{-1}I(s \in \chi)] = 1$. By linearity of expectation we conclude

$$\mathbb{E}_{\Omega_l}[p^{-1}|\chi|] = \sum_{s \in A} \mathbb{E}_{\Omega_l}[p^{-1}I(s \in \chi)] = |A|.$$