Optimal Bounds for Approximate Counting

Jelani Nelson* Huacheng Yu†

March 30, 2022

Abstract

Storing a counter incremented N times would naively consume $O(\log N)$ bits of memory. In 1978 Morris described the very first streaming algorithm: the "Morris Counter" [Mor78]. His algorithm's space bound is a random variable, and it has been shown to be $O(\log\log N + \log(1/\varepsilon) + \log(1/\delta))$ bits in expectation to provide a $(1+\varepsilon)$-approximation with probability $1-\delta$ to the counter's value. We provide a new simple algorithm with a simple analysis showing that randomized space $O(\log\log N + \log(1/\varepsilon) + \log\log(1/\delta))$ bits suffice for the same task, i.e. an exponentially improved dependence on the inverse failure probability. We then provide a new analysis showing that the original Morris Counter itself, after a minor but necessary tweak, actually also enjoys this same improved upper bound. Lastly, we prove a new lower bound for this task showing optimality of our upper bound. We thus completely resolve the asymptotic space complexity of approximate counting. Furthermore all our constants are explicit, and our lower bound and tightest upper bound differ by a multiplicative factor of at most $3 + o(1)$.

*UC Berkeley. minilek@berkeley.edu. Supported by NSF award CCF-1951384, ONR grant N00014-18-1-2562, ONR DORECG award N00014-17-1-2127, an Alfred P. Sloan Research Fellowship, and a Google Faculty Research Award.

†Princeton University. yuhch123@gmail.com.

1 Introduction

Suppose one wishes to maintain an integer N, initialized to zero, subject to a sequence of increment operations. Maintaining this counter exactly can be accomplished using $\lceil \log_2 N \rceil$ bits. In the first example of a non-trivial streaming algorithm, Morris gave a Monte Carlo randomized "approximate counter", which lets one report a constant factor approximation to N with large probability while using $o(\log N)$ bits of memory. His algorithm, the "Morris Counter", uses $O(\log \log N)$ bits [Mor78]. The Morris Counter was later analyzed in more detail [Fla85, GS09], where it was shown that $O(\log \log N + \log(1/\varepsilon) + \log(1/\delta))$ bits of memory is sufficient to return a $(1+\varepsilon)$ approximation with success probability $1-\delta$; the space consumption is a random variable, and this quantity is its expectation (and in fact, the space bound holds with large probability). Further historical details can be found in [Lum18].

Our main contribution is a new, simple improved algorithm and matching lower bound. In particular, we show that the correct dependence on the inverse failure probability is only doubly and not singly logarithmic. This implies for example that $O(\log \log N)$ memory suffices to have failure probability $1/\,poly(N)$, whereas previous Morris Counter analyses only guaranteed failure probability $1/\,poly(\log N)$ in such space.

Theorem 1.1. *For any $\varepsilon, \delta \in (0, 1/2)$ there is a randomized algorithm for approximate counting which outputs $\hat{N}$ satisfying*

$$\mathbb{P}\left(\left|N - \hat{N}\right| > \varepsilon N\right) < \delta. \tag{1}$$

The memory in bits is a random variable M such that for any $S > C(\log \log N + \log(1/\varepsilon) + \log \log(1/\delta))$,[1]

$$\mathbb{P}(M > S) < \exp(-C' \exp(C'' S)). \tag{2}$$

Furthermore, our algorithm is asymptotically optimal up to a constant factor: any randomized algorithm which is promised that the final counter is in the set $\{1, \ldots, n\}$ and which satisfies Eq. (1) must use $\Omega(\min\{\log n, \log \log n + \log(1/\varepsilon) + \log \log(1/\delta)\})$ bits of memory with high probability.

Note the first term in the $\min$ of the lower bound of Theorem 1.1 is matched by a deterministic counter. We further note the space usage of the Morris Counter is also a random variable which satisfies a bound similar to Eq. (2). Next, we show that the Morris Counter itself parametrized to use the same space bound also achieves (1) as long as the counter N is sufficiently large, i.e. at least some value $N_\delta = \Omega(\log(1/\delta))$. This is a mild restriction, since one can simply maintain a deterministic counter in parallel to the Morris Counter up to the value $N_\delta + 1$. Then to answer queries, if the counter is at most N_δ, we return it; else if it equals $N_\delta + 1$, we return the estimator based on the Morris Counter. As we show in the appendix, this minor tweak is necessary; without it, the Morris Counter would not achieve success probability $1-\delta$ in the desired space. We call this slight tweak "Morris+", which is similar to a method used in [GS09]. Our next theorem provides an improved analysis of Morris+. All logarithms in this paper are base 2, unless it is stated otherwise.

Theorem 1.2. *For any $\varepsilon, \delta \in (0, 1/2)$ Morris+, instantiated with appropriate parameters, uses $\log \log N + 2\log(1/\varepsilon) + \log \log(1/\delta) + O(1)$ bits of memory with high probability and outputs $\hat{N}$ satisfying*

$$\mathbb{P}\left(\left|N - \hat{N}\right| > \varepsilon N\right) < \delta. \tag{3}$$

[1] In fact our analysis is more refined and produces explicit constant factors; see Theorem 2.3 and Remark 2.5.

Though we provide two proofs of the same upper bound, we believe both have value. One perhaps pedagogical advantage of Theorem 1.1 is that the new algorithm we provide is designed with the analysis in mind, leading to an overall proof of our novel optimal upper bound that is both short and intuitive. That is, one reads the argument and feels they clearly understand "why" the upper bound is what it is. Meanwhile, the advantage of Theorem 1.2 is that it provides a tight analysis of an algorithm commonly used in practice, albeit at the pedagogical cost that the proof of the theorem boils down to a technical calculation, and the reason the final bound comes out the way it does is arguably less intuitive.

Given that most modern machines have much more than $\log N$ bits of memory for even for N being on the order of the number of particles in the universe, one might wonder whether approximate counting is of real importance or merely a purely intellectual pursuit. An application to keep in mind is not that there is merely one counter, but we may wish to maintain many such counters. In a real such application the number of approximate counters could be very large, and so cutting the number of bits per counter by even a constant factor could be of value. Indeed this was Morris' own original motivation: he needed to keep track of not only one counter, but 26^3 counters, to keep trigram counts as part of the spellchecker typo [Lum18]. An example of a real such scenario in the modern day is the implementation of the "Least Frequently Used" (LFU) cache eviction policy in Redis, one of the most popular in-memory databases. The Redis implementation of this eviction policy needs to keep track of a counter for each key in the database, corresponding to the number of times it has been queried recently. To save memory, these counters are in fact implemented as approximate counters [Red].

This motivating perspective also reveals that typically the memory requirement to calculate the state transition of the approximate counter after an increment, or to answer a query, is much less important; rather, minimizing the memory required to maintain program state is of higher practical relevance, as that affects total storage. Furthermore, if we are maintaining M counters then it is natural to want $\delta \ll 1/M$ so that each counter is approximately correct with high probability. If M is large, then requiring $\log(1/\delta) \geq \log M$ bits per counter may provide no benefit over a naive $\log N$ bit counter for realistic values of N.

In addition to potential practical relevance, from a theoretical perspective "maintaining a counter" is a natural problem and as such the Morris Counter has found applications to other streaming problems. For example, Jayaram and Woodruff showed that for $p \in (0, 1]$ an approximate counter can be used effectively as a subroutine in an algorithm for approximating the pth moment of an insertion-only stream up to $1 + \varepsilon$ in $\tilde{O}(1/\varepsilon^2 + \log n)$ bits of space [JW19], improving over a derandomization of an algorithm of Indyk that uses $O(\varepsilon^{-2} \log n)$ bits [Ind06, KNW10]. Approximate counting also finds use in approximating large frequency moments [AMS99, GS09], approximate reservoir sampling [GS09], approximating the number of inversion when streaming over a permutation [AJKS02], and ℓ_1 heavy hitters in insertion-only streams [BDW19].

1.1 Comparison with previous bounds from [Fla85]

As we discuss in Subsection 1.2, the Morris Counter works by storing a counter X and incrementing it with probability $1/(1 + a)^X$ per update for some parameter $a > 0$. The work [Fla85] characterized the behavior of the Morris algorithm *exactly* when $a = 1$. Unfortunately, the Morris Counter for $a = 1$, which uses $O(\log \log N)$ bits of memory with high probability (which is $O(\log \log N + \log \log(1/\delta))$ for $\delta = 1/\,poly(N)$), does not enjoy constant factor approximation with success probability any better than a constant even for large N, let alone with probability $1 - 1/\,poly(N)$. This failure of the Morris Counter to achieve very high success probability for $a = 1$ is implied by the exact characterization of the algorithm given in [Fla85] itself; Proposition 3 of that work implies that the probability that X fails to be in the interval $[\log_2 N - C, \log_2 N + C]$ equals a constant (depending on C), and X being in that interval is required for the Morris Counter to provide a 2^C-approximation. Thus, the failure probability when $a = 1$ is not even

$o(1)$. Our Theorem 1.2 reveals though that the Morris Counter with $a = \Theta(1/\log N)$ *does* achieve failure probability $1/\,poly(N)$, which is "for free" (up to a constant factor) compared with $a = 1$ since this smaller setting of a still only requires the Morris Counter to use $O(\log\log N)$ bits of memory with high probability.

[Fla85] does have some discussion on using smaller a. Specifically, [Fla85, Section 5] mentions that if one wants error better than the case $a = 1$ to estimate N, one can either average independent counters or change base, and that the former has "an effect similar to" the latter. A variance bound is then given for estimating N when using arbitrary a. This variance bound seems to reveal though that the effects of averaging versus changing base are not similar from a computational complexity perspective: the former requires averaging $\Omega(1/\varepsilon^2)$ copies of the counter, blowing up the space complexity by $1/\varepsilon^2$. The latter leads to a space bound depending only on $O(\log(1/\varepsilon))$. Both yield $O(\log(1/\delta))$ space dependence on the failure probability δ. Equation (46) of [Fla85] does give an explicit sum-product formula for the exact probabilities $P_{n,\ell}$ that the counter exactly equals ℓ after n increments, but this formula is not readily prescriptive for how a should be set in order to achieve relative error $1+\varepsilon$ with failure probability δ.

1.2 Overview of approach

We first explain the idea behind the Morris Counter. The traditional, deterministic and exact counter stores an integer X, initialized to zero. After every increment to N, we increment X with probability 1.0^X, i.e. we always increment it. Thus we can "estimate" N as X, and this estimator has zero variance and is unbiased, at the cost of using $\log N$ memory. Morris instead increments X with probability 0.5^X; this trades off variance for memory. Specifically, one can show that $\mathbb{E}[2^X - 1] = N$, though the variance only satisfies $\mathrm{Var}[2^X - 1] = N(N-1)/2$. A natural idea of Morris is then to change the base of the exponential when deciding the probability to increment X, which turns out to provide a smooth tradeoff between memory and space consumption. Specifically, for any $a > 0$ if incrementing X with probability $1/(1+a)^X$, the expression $a^{-1}((1+a)^X - 1)$ is an unbiased estimator of N with variance $aN(N-1)/2$ (we call the Morris Counter with this parameterization "Morris(a)"). Setting $a = 2\varepsilon^2\delta$, one obtains the guarantee Eq. (1) via Chebyshev's inequality. Note that the space consumption $S := \lceil \log_2 X \rceil$ is a random variable, but is at most $O(\log\log N + \log(1/\varepsilon) + \log(1/\delta))$ with high probability. This is because for $C > 2$, once $X > Z := (\log N/(2\varepsilon^2\delta))^C$, by a union bound the probability that any of the remaining at most N increments causes X to increment even once more is at most $N(1+2\varepsilon^2\delta)^{-Z} < e^{-(\log N/(2\varepsilon^2\delta))^{C-1}} < N^{-\omega(1)}$ (using that $(1-r)^{1/r} < 1/e$ for $r > 0$). Thus, with high probability the Morris Counter uses at most $O(\log Z) = O(\log\log N + \log(1/\varepsilon) + \log(1/\delta))$ bits of memory.

We now describe our new algorithm. First, we consider a promise decision problem: given some $T > 1$ and $\varepsilon \in (0,1)$, decide whether $N < (1-\varepsilon/10)T$ or $N > (1+\varepsilon/10)T$ when promised that one of the two holds. We can solve this decision problem as follows. We store a counter Y in memory, initialized to 0. Set $\alpha = \min\{1, C\log(1/\eta)/(\varepsilon^2 T)\}$ for C a large constant and $\eta \in (0,1)$ a parameter to be set. For each increment to N, if $Y \le \alpha T$ then increment Y with probability α; else do nothing. At query time, we declare $N > (1+\varepsilon/10)T$ iff $Y > \alpha T$. A Chernoff bound shows that this procedure is correct with probability at least $1-\eta$. Furthermore the memory consumed is guaranteed to be $O(\log(\alpha T)) = O(\log(1/\varepsilon) + \log\log(1/\eta))$.

Now to solve the full approximate counting problem, and not just the decision problem, we solve multiple instantiations of the above promise problem in sequence, where in iteration j we use the threshold $T_j = (1+\varepsilon)^j$ and increment probability $\alpha_j = \min\{1, C\log(1/\eta_j)/(\varepsilon^3 T_j)\}$ for $\eta_j < C\delta/j^2$ (chosen so that by a union bound, the probability that we ever fail to solve the promise problem in any iteration j is at most $\sum_j \eta_j \le \delta$). When Y reaches the value $\alpha_j T_j$, we increase j and correspondingly set $Y \leftarrow \lfloor Y \cdot \alpha_{j+1}/\alpha_j \rfloor$ (which is "correct in expectation", since the number of increments we would have done in expectation with

parameter α_{j+1} is an $\alpha_j/\alpha_{j+1} \approx 1+\varepsilon$ factor less). To answer a query for N, we simply return T_j. The adjustment from ε^2 to ε^3 in α_j is for technical reasons (see the proof of Theorem 2.1).

We next provide an improved analysis of Morris' original algorithm. To do so, we define the random variable Z_i to be the number of increments that Morris(a), run for an infinite number of increments, would have its counter X equal to i before incrementing to $X = i+1$. Then Z_i is a geometric random variable with parameter $1/(1+a)^i$, and we are able to show the desired behavior of Morris(a) by proving concentration bounds on prefix sums of the Z_i via analyzing its moment-generating function.

Our new lower bound comes from showing that a randomized approximate counter using space S can be made deterministic with no increased space cost at the cost of increasing its failure probability by factors that grow with S. If S is smaller than a certain threshold (the lower bound we are trying to prove), this argument leads to a correct space-$o(\log n)$ deterministic algorithm for the problem, which is impossible, and thus the space-S algorithm for S so small could not have existed.

1.3 Notation

We use C, C', C'' to denote universal positive constants, which may change from line to line. We also use $A \pm B$ to denote a value in the interval $[A-B, A+B]$, with $D = A \pm B$ signifying $D \in [A-B, A+B]$. As mentioned, we also use "Morris(a)" to refer to the Morris Counter parameterized to increment X with probability $1/(1+a)^X$.

2 Improved upper bound for approximate counting

In Subsection 2.1 we describe and analyze our new algorithm for approximate counting with space complexity $O(\log\log N + \log(1/\varepsilon) + \log\log(1/\delta))$. We then show that this upper bound is achieved by the original Morris Counter itself in Subsection 2.2.

2.1 New algorithm description and analysis

We describe our full approximate counting algorithm in Algorithm 1. The counter is initialized via the **Init()** procedure, and each increment to N and query for an estimate of N are described in the pseudocode, following the ideas set forth in Subsection 1.2. Theorem 2.1 shows that the relative error of the output of Algorithm 1 is $1+O(\varepsilon)$ with probability $1-O(\delta)$. Eq. (1) follows by adjusting ε, δ by a constant factor. Our variable X is quite similar to that of the Morris Counter: it represents (an approximation to) $\log_{1+\varepsilon} N$. The main difference is that whereas the Morris Counter decides to increment X based on flipping a number of coins depending on X itself, we use an auxiliary counter Y to guide when X should be incremented.

First we define some notation that will be useful for the proof. We divide the algorithm's execution into epochs $k = 0, 1, 2, \ldots$, corresponding to the value of $X - X_0$. We mark the end of an epoch immediately before line 8 is about to execute, and the beginning of the new epoch immediately after line 13 has completed executing. During a given epoch, we let T_k, α_k, η_k be the corresponding values of T, α, η set in lines 7–12 of Algorithm 1. For example, $T_0 = 1, \alpha_0 = 1, \eta_0 = \delta$. We also define Y_k to be the value of Y when epoch k begins, so that $Y_0 = 0$ and Y_k for $k > 0$ is set in line 11 of Algorithm 1. To be precise, a particular epoch is said to begin after Algorithm 1 completes lines 4 or 12, and it ends at line 6 when the if statement triggers. We say that N becomes a certain value once **Increment()** has been called that number of times, *and* the most recent call completed.

Algorithm 1 Approximate counting algorithm.

procedure ApproxCount(ε, δ)
  Init():
  $\eta \leftarrow \delta, X_0 \leftarrow \lceil \ln_{1+\varepsilon}(C \ln(1/\eta)/\varepsilon^3) \rceil$
  $Y \leftarrow 0, X \leftarrow X_0, \alpha \leftarrow 1, T \leftarrow \lceil (1+\varepsilon)^X \rceil$

  Increment():
  with probability α, update $Y \leftarrow Y + 1$
  if $Y > \alpha T$ **then**
    $X \leftarrow X + 1$
    $T \leftarrow \lceil (1+\varepsilon)^X \rceil, \eta \leftarrow \frac{\delta}{X^2}$
    $\alpha_{\text{new}} \leftarrow \frac{C \ln(1/\eta)}{\varepsilon^3 T}$
    $Y \leftarrow \lfloor Y \cdot \alpha_{\text{new}}/\alpha \rfloor$
    $\alpha \leftarrow \alpha_{\text{new}}$
  end if

  Query():
  if $X = X_0$ **then**
    return Y
  else
    return T
  end if
end procedure

Theorem 2.1. *There is a universal constant $C' > 0$ such that $\forall \varepsilon, \delta \in (0, 1/2)$, the output $\hat{N}$ of **Query()** in Algorithm 1 satisfies $\mathbb{P}(|\hat{N} - N| > C'\varepsilon N) < C'\delta$.*

Proof. We first note that while remaining in epoch 0, i.e. as long as $1 \le N \le T_0$, Y stores N exactly and thus our output is exactly correct. Our focus is thus on the case of larger N.

For $k \ge 0$, define the event $\mathcal{E}_k$ that once we enter epoch k, the number of increments to N before we advance to the next epoch is $T_k - T_{k-1} \pm \varepsilon^2 T_{k-1}$ (where we use the convention $T_{-1} = 0$). We henceforth condition on the event $\wedge_{k \ge 0} \mathcal{E}_k$. Since the T_r are in geometric series with base $1 + \varepsilon$ (up to ± 1 due to rounding), we have $\sum_{r=0}^{k} (T_r - T_{r-1} \pm \varepsilon^2 T_{r-1}) \subseteq (1 \pm 1.5\varepsilon) T_k$, i.e., only after $(1 \pm 1.5\varepsilon)T_k$ increments to N, could the algorithm possibly be in epoch k. Thus, if k^* is the final epoch when **Query()** is called, we have $\hat{N} = T_{k^*}$ and $N = (1 \pm 1.5\varepsilon) T_{k^*}$. That is, $\hat{N} = \frac{1}{1 \pm 1.5\varepsilon} N$, which implies $|\hat{N} - N| \le C\varepsilon N$ when $\varepsilon < 1/2$.

We finally bound

$$\mathbb{P}\left(\bigwedge_{k=0}^{\infty} \mathcal{E}_k \right) = 1 - \mathbb{P}\left(\bigvee_{k=0}^{\infty} \neg \mathcal{E}_k \right) \ge 1 - \sum_{k=0}^{\infty} \mathbb{P}(\neg \mathcal{E}_k).$$

$\mathbb{P}(\neg \mathcal{E}_0) = 0$, so we focus on $k \ge 1$. Note $Y_k = \lfloor (\lfloor \alpha_{k-1} T_{k-1} \rfloor + 1) \cdot (\alpha_k / \alpha_{k-1}) \rfloor$, which is $\alpha_k T_{k-1} \pm O(1)$. The new threshold for Y to enter epoch $k+1$ is $\lfloor \alpha_k T_k \rfloor + 1$, which thus requires $\alpha_k (T_k - T_{k-1}) \pm O(1)$ more increments to Y, which is

$$\varepsilon \alpha_k T_{k-1} \pm O(1), \tag{4}$$

since $T_k - T_{k-1} = \varepsilon T_{k-1} \pm O(1)$ and $\alpha \leq 1$. To upper bound the probability that we already advance to the next epoch after calling **Increment()** $t_1 := T_k - T_{k-1} - \varepsilon^2 T_{k-1}$ times, it suffices to consider the following question: If we increment Y with probability α_k independently for each of the t_1 **Increment()** calls, what is the probability that we increment Y *at least* $\varepsilon\alpha_k T_{k-1} - O(1)$ times.[2]

The expected number of times Y is incremented is

$$\alpha_k t_1 = \varepsilon\alpha_k T_{k-1} - \varepsilon^2\alpha_k T_{k-1} \pm O(1),$$

which is $\Theta(\ln(1/\eta_k)/\varepsilon^2)$. Advancing to the next epoch thus implies deviating from the expectation by more than $\varepsilon^2\alpha_k T_{k-1} \pm O(1)$, i.e., ε times the expectation. The Chernoff bound implies that the probability of this occurring is at most η_k. A similar calculation shows that the probability that we have *not* advanced to the next epoch after calling **Increment()** $t_2 := T_k - T_{k-1} + \varepsilon^2 T_{k-1}$ times. Thus $\mathbb{P}(\neg\mathcal{E}_k) \leq 2\eta_k$. Thus $\mathbb{P}(\vee_{k\geq 0}\neg\mathcal{E}_k) \leq 2\sum_k \eta_k = 2\sum_k \delta/(k+1)^2 = O(\delta)$. □

Remark 2.2. Before we give the space analysis, the astute reader may notice that T itself is ideally approximately N and thus should require $\Theta(\log N)$ bits to store. A similar statement could be made about the Morris Counter: the output is ultimately given as $a^{-1}((1+a)^X - 1)$ (see Subsection 1.2), which is also $\Theta(\log N)$ bits. The key is that in implementation, we never actually store T: we only store X. Then our answer to a query is only to return X, which will be an additive $O(1)$ approximation to $\log_{1+\varepsilon} N$ with high probability, which is enough for the querying party to specify an approximation to N. Similarly, δ is never stored or even given to the algorithm, but rather the input should be Δ such that $\delta = 2^{-\Delta}$, and only Δ is ever stored. Also, the correctness analysis only requires that α be *at least* the value in line 10 and not exactly that (to apply the Chernoff bound effectively). Thus α can be rounded up to the nearest inverse power of 2 so that $\alpha = 2^{-t}$ and only t need be stored consuming only $\log t = \log\log(1/\alpha)$ bits. We can then generate a $\mathsf{Bernoulli}(\alpha)$ random variable (line 6) by flipping a fair coin t times and returning 1 iff all flips were heads; this takes 1 bit to keep track of the AND and $\log t$ bits to keep track of the number of flips made so far. η also need not be stored explicitly since its value is implicit from other stored values (namely X, ε, and Δ).

Of course the situation is even simpler in models of computation other than word RAM, such as a finite automaton or branching program: then program constants need not be stored in memory (they only affect the transitions), and only the variables X, Y contain program state that needs to be stored. Furthermore, what is most important from the perspective of the practical motivation in Section 1 when running a system storing many approximate counters is the number of bits required to maintain program state; it is reasonable to assume in practical applications that $O(\log N)$ bit registers are available to be used temporarily while processing updates and queries, which could lead to faster and simpler implementation.

Theorem 2.3. *For any $\varepsilon, \delta \in (0, 1/2)$, the probability that Algorithm 1 needs more than*

$$\log\log N + \log\log(1/\delta) + 3\log(1/\varepsilon) + \Omega(t)$$

bits of memory after N increments is at most $(\varepsilon/N)^{2^t}$, for any $t \geq C \cdot (\log\log\log N + \log\log(1/\varepsilon))$, where C is a sufficiently large constant.

To see that this theorem implies the space bound stated in Theorem 1.1, for any $S > C(\log\log N + \log(1/\varepsilon) + \log\log(1/\delta))$ for a sufficiently large C, we have $t > (C-3)(\log\log N + \log(1/\varepsilon) + \log\log(1/\delta)) >$

[2]Note that in the actual execution of the algorithm, not all t_1 calls increment Y with probability α_k, e.g., if we have advanced to the next epoch already, then the probability becomes α_{k+1}. Nevertheless, the probability that we advance to the next epoch after t_1 **Increment()** calls is the same if we increment Y with α_k probability for each call, since it does not matter if we have *already* advanced to the next epoch.

$S/2$. Hence, the probability that we use more than S bits of memory after N increments is at most

$$(\varepsilon/N)^{2^t} \leq 2^{-2^t} \leq \exp(-C' \exp(C'' S)),$$

for some constants $C', C'' > 0$.

Proof. As described in Remark 2.2, Algorithm 1 only explicitly stores two variables X and Y. When $X = X_0$, Y is between 0 and $T = O(\log(1/\delta)/\varepsilon^3)$. In this case, storing Y takes

$$\log\log(1/\delta) + 3\log(1/\varepsilon) + O(1)$$

bits. When $X = X_0 + k$ for $k \geq 1$ (i.e., in epoch k), Y is between $\alpha_k T_{k-1} - O(1)$ and $\alpha_k T_k + O(1)$. In this case, storing Y takes

$$\begin{aligned}\log(\alpha_k(T_k - T_{k-1}) + O(1)) &\leq \log\log(1/\eta) + 2\log(1/\varepsilon) + O(1) \\ &\leq \log\log(1/\delta) + 2\log(1/\varepsilon) + 2\log\log X + O(1)\end{aligned}$$

bits. Thus, provided that $X \leq X_{\max}$, Algorithm 1 uses at most

$$\max\{\log X_{\max}, \log(1/\varepsilon)\} + \log\log(1/\delta) + 2\log(1/\varepsilon) + 2\log\log X_{\max} + O(1) \tag{5}$$

bits. In the following, we show that the final X is small with high probability.

We will show that once we reach an epoch k for k large (corresponding to $X = X_0 + k$), with high probability we will never advance to epoch $k+1$. Indeed, the probability that we do advance is the probability that Y increments at least $\varepsilon\alpha_k T_{k-1} \pm O(1)$ times over the at most N remaining calls to **Increment()** (see Eq. (4)). By a union bound over all $(\varepsilon\alpha_k T_{k-1} + O(1))$-subsets of the remaining increments, the probability that this occurs is at most

$$\begin{aligned}\binom{N}{\varepsilon\alpha_k T_{k-1} \pm O(1)} \cdot \alpha_k^{\varepsilon\alpha_k T_{k-1} \pm O(1)} &\leq \left(\frac{2eN}{\varepsilon T_{k-1}}\right)^{\varepsilon\alpha_k T_{k-1} \pm O(1)} \\ &\leq \left(\frac{C'N}{\varepsilon(1+\varepsilon)^X}\right)^{\Theta(\log(X^2/\delta)/\varepsilon^2)}.\end{aligned}$$

For $X \geq 2\log_{1+\varepsilon}(N/\varepsilon)$, it is at most

$$\begin{aligned}\left(\frac{C'N}{\varepsilon(1+\varepsilon)^X}\right)^{\Theta(\log(X^2/\delta)/\varepsilon^2)} &\leq \left(\frac{1}{(1+\varepsilon)^X}\right)^{\Omega(1/\varepsilon^2)} \\ &\leq (e^{-\Theta(\varepsilon X)})^{\Omega(1/\varepsilon^2)} \\ &\leq e^{-\Omega(X)}.\end{aligned}$$

By setting $X_{\max} = \Theta(2^t \log_{1+\varepsilon}(N/\varepsilon))$ for some integer $t \geq C \cdot (\log\log\log N + \log\log(1/\varepsilon))$, i.e., $t \geq \log\log X_{\max} + \log\log(1/\varepsilon)$,

$$\begin{aligned}\log X_{\max} &\leq \log\log N + \log(1/\varepsilon) + \log\log(1/\varepsilon) + t + O(1) \\ &= \log\log N + \log(1/\varepsilon) + \Theta(t).\end{aligned}$$

By Equation (5), the probability that Algorithm 1 needs more than

$$\log\log N + \log\log(1/\delta) + 3\log(1/\varepsilon) + \Omega(t)$$

bits of space is at most

$$\left(\frac{\varepsilon}{N}\right)^{2^t}.$$

□

Remark 2.4. In the proof, we assumed that the algorithm allocates exactly $\log X_{\max}$ bits to store X, and then we bounded the probability that X exceeds $X_{\max}$ after N increments. This assumption requires us to have an upper bound on N in advance. In general, when an upper bound on N is unknown, we will have to store variable X that is also unbounded, and dynamically allocate bits to the counter. This can be done by first encoding $\lceil \log X \rceil$ using $O(\log \log X)$ bits, then encoding X using $\lceil \log X \rceil$ bits. Our proof gives the same space bound in this case.

Remark 2.5. The source of the constant factor "3" multiplying $\log(1/\varepsilon)$ in the space complexity is due to the cubic dependence of α_{new} on $1/\varepsilon$ in Algorithm 1. This cubic dependence was due to the proof structure of Theorem 2.1: we conditioned on the events $\mathcal{E}_k$ that we spent a concentrated amount of time in each epoch. To show that this happens with high probability, we performed a union bound over all epochs. We feel this structure makes the proof more intuitive, though it comes at the cost of a worsened constant factor. One can show that the algorithm is still in fact correct with α_{new} depending only quadratically on $1/\varepsilon$ by proving concentration only on the total time spent on all the epochs combined, as opposed to union bounding over epochs separately, by using an argument similar to what we will see shortly in Subsection 2.2. One can also see empirically via implementation that the algorithm of this section and Morris+ behave nearly identically, including the constant factor (see Section 4).

Remark 2.6. Our approximate counter is fully mergeable [ACH+13]. That is, given two counters (X_1, Y_1) and (X_2, Y_2), which approximate two (unknown) numbers N_1 and N_2 respectively, they can be merged into a single data structure (X, Y) that follows the same distribution as if it was incremented exactly $N_1 + N_2$ times so that nothing is lost in the parameters ε and δ (the Morris Counter enjoys this same benefit [CY20, Section 2.1]). To see this, observe that each epoch of our algorithm uses sampling, and the sampling rate is non-increasing. Assuming $X_1 \le X_2$, we can simulate N_1 extra increments to the second counter by another subsampling with the correct probabilities. More specifically, the first counter is in epoch $k_1 = X_1 - X_0$, and we know the sampling probabilities $\alpha_0, \ldots, \alpha_{k_1}$, and the exact number of increments that survived the sampling (caused Y_1 to increment) in each epoch. We are going to insert all the survivors to the second counter, which currently have sampling probability α_{k_2} for $k_2 = X_2 - X_0$. For each survivor in epoch i (for $0 \le i \le k_1$), we increment Y_2 with probability α_{k_2}/α_i. Then effectively, we increment Y_2 with probability α_{k_2} for each of the *original* N_1 increments. Whenever Y_2 reaches the threshold αT, we increment X_2, update Y_2, and adjust the probabilities. Hence, the final (X_2, Y_2) has the same distribution as if it was incremented a total of $N_1 + N_2$ times.

2.2 Morris Counter improved analysis

Here we analyze the Morris(a) algorithm for some $a \in (0, 1)$, in which X is incremented with probability $(1+a)^{-X}$ and we output $\hat{N} = ((1+a)^X - 1)/a$. When the total number of increments N is at most $8/a$, the value of the counter can be explicitly maintained in addition to the Morris Counter, which costs at most $\log(1/a) + O(1)$ bits of space. In the following, we assume N is at least $8/a$; this is not a serious limitation since we can maintain a separate counter exactly, deterministically up until this value (the "Morris+" modification described in Section 1).

Let us consider Morris(a) on an infinite sequence of increments. For any $i \geq 0$, X exceeds i with probability 1. Let $Z_i \geq 1$ be the random variable denoting the number of increments it takes for X to increase from i to $i+1$. Since when $X = i$, each increment causes X to increase with probability $p_i = (1+a)^{-i}$, Z_i follows the geometric distribution

$$\mathbb{P}[Z_i = l] = (1-p_i)^{l-1} p_i.$$

Therefore, we have

$$\mathbb{E}[Z_i] = 1/p_i = (1+a)^i,$$

and

$$\mathbb{E}[e^{tZ_i}] = \sum_{l \geq 1} e^{tl}(1-p_i)^{l-1} p_i = \frac{e^t p_i}{1 - e^t(1-p_i)},$$

for any t such that $e^t(1-p_i) < 1$.

Next, let $\varepsilon < 1/2$, we bound

$$\mathbb{P}\left[\sum_{i=0}^{k} Z_i \geq (1+\varepsilon)\sum_{i=0}^{k} 1/p_i\right]. \tag{6}$$

Following the proof of Chernoff bound, for t such that $e^t(1-p_k) < 1$, we have

$$\begin{aligned}
\mathbb{E}\left[e^{t\sum_{i=0}^{k} Z_i}\right] &= \prod_{i=0}^{k} \mathbb{E}\left[e^{tZ_i}\right] \\
&= \frac{e^{(k+1)t}\prod_{i=0}^{k} p_i}{\prod_{i=0}^{k}(1 - e^t(1-p_i))} \\
&= \frac{e^{(k+1)t}(1+a)^{-k(k+1)/2}}{\prod_{i=0}^{k}(1 - e^t(1-p_i))}.
\end{aligned}$$

By Markov's inequality,

$$\begin{aligned}
(6) &\leq \frac{\mathbb{E}\left[e^{t\sum_{i=0}^{k} Z_i}\right]}{e^{t(1+\varepsilon)\sum_{i=0}^{k} 1/p_i}} \\
&= \frac{\mathbb{E}\left[e^{t\sum_{i=0}^{k} Z_i}\right]}{e^{t(1+\varepsilon)((1+a)^{k+1}-1)/a}} \\
&= \frac{e^{(k+1)t}(1+a)^{-k(k+1)/2}}{e^{t(1+\varepsilon)((1+a)^{k+1}-1)/a}\prod_{i=0}^{k}(1 - e^t(1-(1+a)^{-i}))}.
\end{aligned}$$

Now set $t = \ln\left(\frac{1}{1-\frac{1}{2}\varepsilon(1+a)^{-k}}\right)$, which satisfies $e^t(1-p_k) < 1$, we have

$$\begin{aligned}
(6) &\le (1+a)^{-k(k+1)/2} \cdot \left(1 - \frac{1}{2}\varepsilon(1+a)^{-k}\right)^{-(k+1)+(1+\varepsilon)((1+a)^{k+1}-1)/a} \cdot \prod_{i=0}^{k} \frac{1}{1-\frac{1-(1+a)^{-i}}{1-\frac{1}{2}\varepsilon(1+a)^{-k}}} \\
&= (1+a)^{-k(k+1)/2} \cdot \left(1 - \frac{1}{2}\varepsilon(1+a)^{-k}\right)^{-(k+1)+(1+\varepsilon)((1+a)^{k+1}-1)/a} \cdot \prod_{i=0}^{k} \frac{1-\frac{1}{2}\varepsilon(1+a)^{-k}}{(1+a)^{-i}-\frac{1}{2}\varepsilon(1+a)^{-k}} \\
&= (1+a)^{-k(k+1)/2} \cdot \left(1 - \frac{1}{2}\varepsilon(1+a)^{-k}\right)^{(1+\varepsilon)((1+a)^{k+1}-1)/a} \cdot \prod_{i=0}^{k} \frac{1}{(1+a)^{-i}(1-\frac{1}{2}\varepsilon(1+a)^{-k+i})} \\
&\le e^{-\frac{1}{2}\varepsilon(1+a)^{-k}(1+\varepsilon)((1+a)^{k+1}-1)/a} \cdot \prod_{i=0}^{k} \frac{1}{1-\frac{1}{2}\varepsilon(1+a)^{-k+i}}.
\end{aligned}$$

By the fact that $1/(1-z) \le e^{z+z^2}$ for all $0 < z < 1/2$,

$$\begin{aligned}
(6) &\le e^{-\frac{1}{2}\varepsilon(1+a)^{-k}(1+\varepsilon)((1+a)^{k+1}-1)/a} \cdot e^{\sum_{i=0}^{k}(\frac{1}{2}\varepsilon(1+a)^{-k+i}+\frac{1}{4}\varepsilon^2(1+a)^{-2k+2i})} \\
&= e^{-\frac{1}{2}\varepsilon(1+a)^{-k}\left((1+\varepsilon)((1+a)^{k+1}-1)/a-\sum_{i=0}^{k}((1+a)^i+\frac{1}{2}\varepsilon(1+a)^{-k+2i})\right)} \\
&\le e^{-\frac{1}{2}\varepsilon(1+a)^{-k}\left((1+\varepsilon)((1+a)^{k+1}-1)/a-(1+\frac{1}{2}\varepsilon)((1+a)^{k+1}-1)/a)\right)} \\
&= e^{-\frac{1}{4}\varepsilon^2(1+a)^{-k}((1+a)^{k+1}-1)/a}.
\end{aligned}$$

For $k > \frac{1}{a}$, we have (6) $\le e^{-\varepsilon^2/8a}$.

Similarly, we next bound

$$\mathbb{P}\left[\sum_{i=0}^{k} Z_i \le (1-\varepsilon)\sum_{i=0}^{k} 1/p_i\right]. \tag{7}$$

By Markov's inequality,

$$\begin{aligned}
(7) &= \mathbb{P}\left[e^{-t\sum_{i=0}^{k} Z_i} \ge e^{-t(1-\varepsilon)\sum_{i=0}^{k} 1/p_i}\right] \\
&\le \frac{\mathbb{E}\left[e^{-t\sum_{i=0}^{k} Z_i}\right]}{e^{-t(1-\varepsilon)\sum_{i=0}^{k} 1/p_i}} \\
&= \frac{e^{-t(k+1)}(1+a)^{-k(k+1)/2}}{e^{-t(1-\varepsilon)((1+a)^{k+1}-1)/a}\prod_{i=0}^{k}(1-e^{-t}(1-p_i))}.
\end{aligned}$$

Now set $t = \ln(1 + \frac{1}{2}\varepsilon(1+a)^{-k})$, we have

$$\begin{aligned}
(7) &\le (1+a)^{-k(k+1)/2} \cdot \left(1 + \frac{1}{2}\varepsilon(1+a)^{-k}\right)^{-(k+1)+(1-\varepsilon)((1+a)^{k+1}-1)/a} \cdot \frac{1}{\prod_{i=0}^{k}\left(1 - \frac{1-(1+a)^{-i}}{1+\frac{1}{2}\varepsilon(1+a)^{-k}}\right)} \\
&= (1+a)^{-k(k+1)/2} \cdot \left(1 + \frac{1}{2}\varepsilon(1+a)^{-k}\right)^{-(k+1)+(1-\varepsilon)((1+a)^{k+1}-1)/a} \cdot \frac{1}{\prod_{i=0}^{k}\left(\frac{(1+a)^{-i}+\frac{1}{2}\varepsilon(1+a)^{-k}}{1+\frac{1}{2}\varepsilon(1+a)^{-k}}\right)} \\
&= (1+a)^{-k(k+1)/2} \cdot \left(1 + \frac{1}{2}\varepsilon(1+a)^{-k}\right)^{(1-\varepsilon)((1+a)^{k+1}-1)/a} \cdot \frac{1}{\prod_{i=0}^{k}(1+a)^{-i}\left(1+\frac{1}{2}\varepsilon(1+a)^{-k+i}\right)} \\
&= \left(1 + \frac{1}{2}\varepsilon(1+a)^{-k}\right)^{(1-\varepsilon)((1+a)^{k+1}-1)/a} \cdot \frac{1}{\prod_{i=0}^{k}\left(1+\frac{1}{2}\varepsilon(1+a)^{-k+i}\right)}.
\end{aligned}$$

By the fact that $1/(1+z) \le e^{-z+z^2}$ for $z \ge 0$, we have

$$\begin{aligned}
(7) &\le e^{\frac{1}{2}\varepsilon(1+a)^{-k}(1-\varepsilon)((1+a)^{k+1}-1)/a} \cdot e^{\prod_{i=0}^{k}\left(-\frac{1}{2}\varepsilon(1+a)^{-k+i}+\frac{1}{4}\varepsilon^2(1+a)^{-2k+2i}\right)} \\
&= e^{\frac{1}{2}\varepsilon(1+a)^{-k}\left((1-\varepsilon)((1+a)^{k+1}-1)/a+\prod_{i=0}^{k}\left(-(1+a)^{i}+\frac{1}{2}\varepsilon(1+a)^{-k+2i}\right)\right)} \\
&\le e^{-\frac{1}{4}\varepsilon^2(1+a)^{-k}((1+a)^{k+1}-1)/a}.
\end{aligned}$$

When $k > \frac{1}{a}$, this is at most $e^{-\varepsilon^2/8a}$.

Therefore, for any $k > 1/a$, with probability at least $1 - e^{-\varepsilon^2/8a}$, we have

$$\left|\sum_{i=0}^{k} Z_i - ((1+a)^{k+1}-1)/a\right| \le \varepsilon((1+a)^{k+1}-1)/a.$$

Now fix any $N > 8/a$, let k_1 be the largest k such that $(1+\varepsilon)((1+a)^{k+1}-1)/a < N$, k_2 be the smallest k such that $(1-\varepsilon)((1+a)^{k+1}-1)/a \ge N$. We have $k_1, k_2 > 1/a$, then we apply the above inequality to k_1 and k_2, and by union bound, with probability at least $1 - 2e^{-\varepsilon^2/8a}$, we have both

$$\sum_{i=0}^{k_1} Z_i \le (1+\varepsilon)((1+a)^{k_1+1}-1)/a < N,$$

i.e., $X > k_1$ after N increments, and

$$\sum_{i=0}^{k_2} Z_i \ge (1-\varepsilon)((1+a)^{k_2+1}-1)/a \ge N,$$

i.e., $X \le k_2$ after N increments. Therefore, $((1+a)^X - 1)/a$ is a $(1 \pm 2\varepsilon)$ approximation of N with probability $1 - 2e^{-\varepsilon^2/8a}$.

By setting $a = \varepsilon^2/(8\ln(1/\delta))$, the space usage of Morris(a) is $\log\log N + \log(1/a) + O(1) = \log\log N + 2\log(1/\varepsilon) + \log\log(1/\delta) + O(1)$ bits with high probability, and outputs a $(1\pm2\varepsilon)$ approximation with probability $1 - 2/\delta$. By reparametrizing, we prove Theorem 1.2.

Remark 2.7. While it may be possible to improve the constant factor "8" in the exponent of the tail bound above, note that this constant in turn only affects the setting of a by a constant factor, and the space complexity of Morris(a) only depends logarithmically on $1/a$. Thus, any improvement to the factor 8 can only improve the analysis of the space complexity by an additive constant.

Remark 2.8. After seeing our proof, Eric Price pointed out that it can be made even more succinct as follows: one can show that geometric random variables are "subgamma", so that a sum of geometric random variables (as in Eqs. (6) and (7)) is subgamma with appropriate parameters (see [BLM13, Section 2.4] for the definition and relevant properties of subgamma random variables).

3 Space lower bound

Here we prove the matching lower bound for approximate counters. Our lower bound states that even if the algorithm's memory usage is a random variable which only has a *small* chance of being small (i.e. we allow it to use arbitrarily large memory with large probability $1 - \sqrt{\delta}$), it still cannot satisfy Eq. (1).

Theorem 3.1. *Fix $\varepsilon, \delta \in (0, 1/2)$ and integer n. Let $\mathcal{C}$ be an approximate counter which outputs $\hat{N}$ satisfying*

$$\mathbb{P}(|N - \hat{N}| > \varepsilon N) < \delta,$$

for all $N \in \{1, \dots, n\}$, and uses no more than S bits of space with probability at least $\sqrt{\delta}$. We must have

$$S \geq \min\{\log n - O(1), \max\{\log\log n + \log(1/\varepsilon) - O(\log\log(1/\varepsilon)), \log\log(1/\delta) - O(\log\log\log(1/\delta))\},$$

which is at least $\Omega(\min\{\log n, \log\log n + \log(1/\varepsilon) + \log\log(1/\delta)\})$.

The first observation is that conditioned on using no more than S bits of space, we have

$$\mathbb{P}(|N - \hat{N}| > \varepsilon N \mid \text{use at most } S \text{ bits}) < \delta / \mathbb{P}(\text{use at most } S \text{ bits}) \leq \sqrt{\delta}.$$

Hence, we may assume that $\mathcal{C}$ always uses at most S bits of space, at the cost of increasing the failure probability to $\sqrt{\delta}$, which is inconsequential since the dependence on δ in the space bound is $\log\log(1/\delta)$. In the following, we assume that $\mathcal{C}$ never uses more than S bits.

Let $T = \lfloor \min\{n/4, \frac{\log(1/\delta)}{4\log\log(1/\delta)}\} \rfloor$. Then for every $N = 1, \dots, T/2$, $\mathcal{C}$ outputs $\hat{N}$ that is less than T with probability $1-\delta$, and for every $N = 2T, 2T+1, \dots, 4T$, $\mathcal{C}$ outputs $\hat{N}$ that is at least T with probability $1-\delta$. In particular, $\mathcal{C}$ distinguishes $N \in [1, T/2]$ and $N \in [2T, 4T]$ with probability $1-\delta$. In the following, we show that any $\mathcal{C}$ that distinguishes the two cases with probability $1 - \delta$ must use $\log T - O(1)$ bits of space. We assume for contradiction that $S \leq \log(T/4)$.

First, let us consider the following "derandomization" of $\mathcal{C}$. $\mathcal{C}$ uses no more than S bits of space, hence, it has at most 2^S different memory states. When **Init()** is called, the algorithm generates a (possibly random) initial memory state. Each time **Increment()** is called, the algorithm examines the current state and updates the memory to a possibly different state (and possibly randomly). Let the "deterministic" version of the algorithm $\mathcal{C}_{\text{det}}$ have the same query algorithm as $\mathcal{C}$, but when **Init()** or **Increment()** is called, it examines the current state and the distribution of the new state (or the initial state) according to $\mathcal{C}$; instead of updating the memory according to this distribution, $\mathcal{C}_{\text{det}}$ always updates it to the state with the *highest* probability in this distribution (in case of tie, pick the lexicographically smallest).

Now let us analyze the error probability of $\mathcal{C}_{\mathrm{det}}$. The initialization and increment algorithms are called exactly $N+1$ times in total. Since $\mathcal{C}_{\mathrm{det}}$ picks the state with the highest probability each time, which has probability at least 2^{-S}, the probability that the execution of $\mathcal{C}$ follows the exact same path as $\mathcal{C}_{\mathrm{det}}$ is at least

$$\left(2^{-S}\right)^{N+1}.$$

Therefore, conditioned on the execution of $\mathcal{C}$ following the same path, its error probability is at most

$$\delta \cdot \left(2^{S}\right)^{N+1}.$$

When $N \le 4T$, it is at most

$$\delta \cdot (T/4)^{4T+1} \le \delta \cdot (\log(1/\delta)/(16\log\log(1/\delta)))^{\log(1/\delta)/\log\log(1/\delta)+1} < 1/3.$$

That is, the error probability of $\mathcal{C}_{\mathrm{det}}$ is at most $1/3$, for every $N \in [1, T/2] \cup [2T, 4T]$.

On the other hand, since both initialization and increment algorithms are deterministic, we may apply an argument similar to the "pumping lemma" for DFAs. Since $2^S \le T/4$, there exists $1 \le N_1 < N_2 \le T/2$ such that $\mathcal{C}_{\mathrm{det}}$ reaches the same memory state after N_1 or N_2 increments. Again by the fact that the increment algorithm is deterministic, $\mathcal{C}_{\mathrm{det}}$ must reach the same memory state after $N_1 + k(N_2 - N_1)$ increments, for all integer $k \ge 0$. In particular, there exists $N_3 \in [2T, 4T]$ such that $\mathcal{C}_{\mathrm{det}}$ reaches this memory state after N_3 increments. However, by the assumption of the algorithm, the query algorithm distinguishes between N_1 increments and N_3 increments with probability at least $2/3$, which is impossible as the algorithm reaches the same memory state in the two cases. This proves that $S \ge \log T - O(1)$, i.e.,

$$S \ge \min\{\log n - O(1), \log\log(1/\delta) - O(\log\log\log(1/\delta)))\}. \tag{8}$$

Finally, we show that $S \ge \min\{\log n, \log\log n + \log(1/\varepsilon)\} - O(1)$ as long as $\delta \in (0, \sqrt{1/2})$. Let $N_j = \left\lceil (e^{16\varepsilon j} - 1)/\varepsilon \right\rceil$, and consider incrementing the counter N_j times for an unknown j. Observe that for $j \ge 0$, we have

$$\begin{aligned}
(1-\varepsilon)N_{j+1} - (1+\varepsilon)N_j &\ge (1-\varepsilon)(e^{16\varepsilon(j+1)} - 1)/\varepsilon - (1+\varepsilon)(e^{16\varepsilon j} - 1)/\varepsilon - (1+\varepsilon) \\
&= ((1-\varepsilon)e^{16\varepsilon} - (1+\varepsilon))e^{16\varepsilon j}/\varepsilon - (3+\varepsilon) \\
&\ge ((1-\varepsilon)(1+16\varepsilon) - (1+\varepsilon))/\varepsilon - (3+\varepsilon) \\
&= 11 - 17\varepsilon \\
&> 0.
\end{aligned}$$

Therefore, for every $j \ge 0$ and $j \le (1/16\varepsilon)\ln(\varepsilon n + 1)$ (hence, $N_j \le n$), $\mathcal{C}$ recovers j with probability $1 - \delta > 1/5$, if the counter is incremented N_j times. By fixing the random bits used by $\mathcal{C}$, at least $1/5$ fraction of such j is successfully recovered. The algorithm must reach a different final state for all such j, implying that

$$2^S \ge \frac{1}{5} \cdot (1/16\varepsilon)\ln(\varepsilon n + 1) = \Omega((1/\varepsilon)\log(\varepsilon n + 1)).$$

When $\varepsilon < 1/n$, it is $\Omega((1/\varepsilon)(\varepsilon n)) = \Omega(n)$, and

$$S \ge \log n - O(1).$$

When $1/n \le \varepsilon < 1/\sqrt{n}$, we have

$$S \ge \log(1/\varepsilon) - O(1) \ge \log(1/\varepsilon) + \log\log n - O(\log\log(1/\varepsilon)).$$

When $\varepsilon \geq 1/\sqrt{n}$, we have

$$S \geq \log(1/\varepsilon) + \log\log(\varepsilon n) - O(1) \geq \log(1/\varepsilon) + \log\log n - O(1).$$

In all three cases, the bounds imply

$$S \geq \min\{\log n - O(1), \log\log n + \log(1/\varepsilon) - O(\log\log(1/\varepsilon))\}. \tag{9}$$

Finally, by (8) and (9), we conclude that

$$\begin{aligned} S &\geq \min\{\log n - O(1), \max\{\log\log n + \log(1/\varepsilon) - O(\log\log(1/\varepsilon)), \log\log(1/\delta) - O(\log\log\log(1/\delta))\} \\ &= \Omega(\min\{\log n, \log\log n + \log(1/\varepsilon) + \log\log(1/\delta)\}). \end{aligned}$$

proving the claimed lower bound.

4 Philosophical digression: the value of implementation

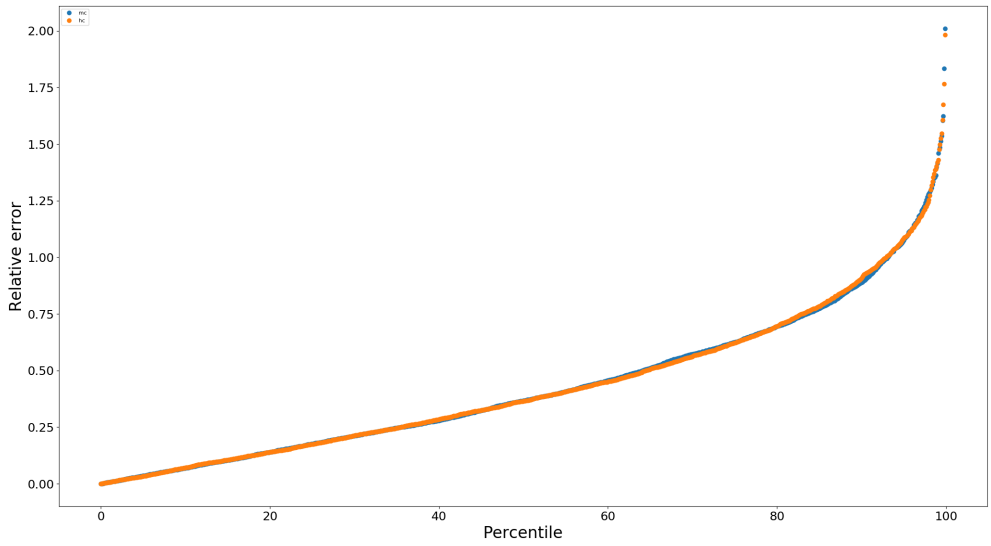

Figure 1: Results of experimental comparison of the Morris counter and a simplified version of the algorithm of Subsection 2.1.

We share in this section a historical note on the development of this work, which may serve the reader as evidence of the value of implementation. Chronologically, we first developed and analyzed the algorithm of Subsection 2.1 and proved the lower bound in Section 3. In the days afterward, excited by the prospect of having a new and improved algorithm for such a fundamental problem, we implemented the Morris Counter as well as (a simplified version of) the algorithm of Subsection 2.1 (and this simplified algorithm is itself similar to the algorithm of [Csu10]) to compare. We ran several experiments. In one, we did the

following 5,000 times for each algorithm, parameterized to use only 17 bits of memory: pick a uniformly random integer $N \in [500000, 999999]$ (thus a 20-bit number) and perform N increments. The results of this experiment are in Fig. 1. The orange plot represents our algorithm, and the blue plot is the Morris Counter. For each respective algorithm's color, a dot plotted at point (x, y) means that in $x\%$ of the trial runs (out of 5,000), the relative multiplicative error of the algorithm's estimate was $y\%$ or less. In other words, we plotted the empirical CDFs of the relative errors of each algorithm. For example, the plot indicates that neither algorithm ever had relative error more than 2.37% in 5,000 runs. The experimental results are plainly apparent: the two algorithms' empirical performances are *nearly identical*! Witnessing this plot convinced us that the previously known analyses of the Morris Counter, an algorithm that has been known for over 40 years and taught in numerous courses, were most likely suboptimal and that the Morris Counter itself is most likely an optimal algorithm for the problem. With the confidence gained from the experimental results, we sought a new and improved analysis of the Morris Counter and succeeded. Thus it seems from this anecdote, implementation can sometimes be valuable even for purely theoretical work.

Acknowledgments

We thank Eric Price for pointing out the content of Remark 2.8 and allowing us to include it here.

References

[ACH+13] Pankaj K. Agarwal, Graham Cormode, Zengfeng Huang, Jeff M. Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. *ACM Trans. Database Syst.*, 38(4):26:1–26:28, 2013.

[AJKS02] Miklós Ajtai, T. S. Jayram, Ravi Kumar, and D. Sivakumar. Approximate counting of inversions in a data stream. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 370–379, 2002.

[AMS99] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[BDW19] Arnab Bhattacharyya, Palash Dey, and David P. Woodruff. An optimal algorithm for ℓ_1-heavy hitters in insertion streams and related problems. *ACM Trans. Algorithms*, 15(1):2:1–2:27, 2019.

[BLM13] Stephane Boucheron, Gabor Lugosi, and Pascal Massart. *Concentration Inequalities: A Nonasymptotic Theory of Independence*. Oxford University Press, 2013.

[Csu10] Miklós Csurös. Approximate counting with a floating-point counter. In *Proceedings of the 16th Annual International Conference on Computing and Combinatorics (COCOON)*, pages 358–367, 2010.

[CY20] Graham Cormode and Ke Yi. *Small summaries for big data (draft)*. 2020. `http://dimacs.rutgers.edu/~graham/ssbd.html`.

[Fla85] Philippe Flajolet. Approximate counting: A detailed analysis. *BIT Comput. Sci. Sect.*, 25(1):113–134, 1985.

[GS09] André Gronemeier and Martin Sauerhoff. Applying approximate counting for computing the frequency moments of long data streams. *Theory Comput. Syst.*, 44(3):332–348, 2009.

[Ind06] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006.

[JW19] Rajesh Jayaram and David P. Woodruff. Towards optimal moment estimation in streaming and distributed models. In *APPROX*, pages 29:1–29:21, 2019.

[KNW10] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1161–1178, 2010.

[Lum18] Jérémie O. Lumbroso. The story of HyperLogLog: How Flajolet processed streams with coin flips. *CoRR*, abs/1805.00612v2, 2018.

[Mor78] Robert H. Morris. Counting large numbers of events in small registers. *Commun. ACM*, 21(10):840–842, 1978.

[Red] Using Redis as an LRU cache. `https://redis.io/topics/lru-cache`. Last accessed Oct 22, 2020.

A Tweaking the Morris Counter is necessary

In this section we show that the modification from the vanilla Morris Counter to "Morris+" described in Section 1 is necessary. Recall the modification: when using Morris(a), we maintain a deterministic counter X' in parallel. During increments, we process the increment both by Morris(a) and by deterministically incrementing X', unless its value is $N_a + 1$ in which case we do not alter it. During queries, if $X' \leq N_a$, we return X'; otherwise we return the estimator from Morris(a) based on X. We set $N_a = 8/a$, as suggested by the analysis in Subsection 2.2.

We now show that if one *does not* modify the Morris Counter but simply uses Morris(a) for $a = \varepsilon^2/(8\ln(1/\delta))$ as suggested in Subsection 2.2, then when $\delta < \varepsilon^{8/3}c^2/16$, $\varepsilon < 1/4$ and the counter value equals $N = N'_a := c\varepsilon^{4/3}/a \geq 2$ for a constant $c \leq 2^{-8}$, the probability that the Morris Counter outputs an estimator $\hat{N} < (1-\varepsilon)N$ is much larger than δ. Note that our analysis requires switching from a deterministic counter to the Morris Counter when $N \geq \Omega(1/a)$ and not $\Omega(\varepsilon^{4/3}/a)$, but the impact on memory complexity is at most a factor of three (and less as N grows): using a deterministic counter up until $N = r$ requires an additional $\lceil \log_2 r \rceil$ bits. Thus the difference between $r = c_1/a$ versus $r = c_2\varepsilon^{4/3}/a$ is the difference between $\log r = \log(c_1) + 3 + \log\log(1/\delta) + 2\log(1/\varepsilon)$ versus $\log r = \log(c_2) + \log\log(1/\delta) + \frac{2}{3}\log(1/\varepsilon)$; i.e. the dependence on $\log(1/\varepsilon)$ differs by a factor of three. Thus our analysis here shows that for small δ, our choice of transition point $r = 8/a$ from a deterministic counter to using the Morris Counter is almost optimal, up to affecting the memory by a multiplicative factor of at most three.

We now show why Morris(a) will fail with probability much larger than δ. Consider the event $\mathcal{E}$ that the Morris Counter increments X in the first t increment operations, and its value remains equal to t in the last $N-t$ increments, for $t = \lfloor \ln(1+(1-2\varepsilon)\varepsilon^{4/3}c)/\ln(1+a) \rfloor$. Recall the estimator is $\hat{N} = a^{-1}((1+a)^X - 1)$.

Thus conditioned on $\mathcal{E}$,

$$\begin{aligned}\hat{N} &= \frac{1}{a}\cdot\left((1+a)^t-1\right)\\ &\le \frac{1}{a}\cdot\left(1+(1-2\varepsilon)\varepsilon^{4/3}c-1\right)\\ &= (1-2\varepsilon)N\\ &< (1-\varepsilon)N\end{aligned}$$

On the other hand, note that $t \ge \ln(1+(1-2\varepsilon)\varepsilon^{4/3}c)/\ln(1+a)-1 \ge \frac{1}{a}\ln(1+(1-2\varepsilon)\varepsilon^{4/3}c)-1$, and $t \le N$. The probability of $\mathcal{E}$ is at least

$$\begin{aligned}\mathbb{P}[\mathcal{E}] &= \prod_{i=0}^{t-1}(1+a)^{-i}\cdot\left(1-(1+a)^{-t}\right)^{N-t}\\ &\ge (1+a)^{-t^2}\cdot\left(1-(1+a)\left(1+(1-2\varepsilon)\varepsilon^{4/3}c\right)^{-1}\right)^{N-\frac{1}{a}\ln(1+(1-2\varepsilon)\varepsilon^{4/3}c)+1}\\ &= (1+a)^{-t^2}\cdot\left(\frac{1+(1-2\varepsilon)\varepsilon^{4/3}c-(1+a)}{1+(1-2\varepsilon)\varepsilon^{4/3}c}\right)^{\frac{1}{a}\left(c\varepsilon^{4/3}-\ln(1+(1-2\varepsilon)\varepsilon^{4/3}c)\right)+1}\\ &\ge (1+a)^{-N^2}\cdot\left(\frac{\varepsilon^{4/3}c}{4}\right)^{\frac{1}{a}\left(c\varepsilon^{4/3}-\ln(1+(1-2\varepsilon)\varepsilon^{4/3}c)\right)+1},\end{aligned}$$

which by the fact that $\ln(1+x)\le x$ and $\ln(1+x)\ge x-x^2/2$ for $x<1$, is

$$\begin{aligned}&\ge \frac{\varepsilon^{4/3}c}{4}\cdot e^{-aN^2}\cdot\left(\frac{\varepsilon^{4/3}c}{4}\right)^{\frac{1}{a}\left(c\varepsilon^{4/3}-(1-2\varepsilon)\varepsilon^{4/3}c+((1-2\varepsilon)\varepsilon^{4/3}c)^2/2\right)}\\ &= \frac{\varepsilon^{4/3}c}{4}\cdot e^{-\frac{1}{a}(\varepsilon^{4/3}c)^2}\cdot e^{-\frac{\ln(4/(\varepsilon^{4/3}c))}{a}\left(2\varepsilon^{7/3}c+\varepsilon^{8/3}c^2/2\right)}\\ &\ge \frac{\varepsilon^{4/3}c}{4}\cdot e^{-\frac{\varepsilon^2}{32a}}\cdot e^{-\frac{\varepsilon^2}{a}\left(4\varepsilon^{1/3}c\ln(4/(\varepsilon^{4/3}c))\right)}\\ &\ge \frac{\varepsilon^{4/3}c}{4}\cdot e^{-\frac{\varepsilon^2}{16a}}\\ &= \frac{\varepsilon^{4/3}c}{4}\cdot\sqrt{\delta}.\end{aligned}$$

When $\delta < \varepsilon^{8/3}c^2/16$, this is larger than δ. Therefore, Morris(a) fails to provide a $(1-\varepsilon)$-approximation for N with probability at least δ.