MORRIS DECAY: OPTIMALITY OF APPROXIMATE COUNTING ALGORITHMS WITH ϵ DECAY

Charles Zhu
Department of Electrical and Computer Engineering
New York University
Brooklyn, NY 11201, USA
cwz2014@nyu.edu

Chenyu Gu
Department of Computer Science and Engineering
New York University
Brooklyn, NY 11201, USA
cg4053@nyu.edu

Minghao Shao
Department of Computer Science and Engineering
New York University
Brooklyn, NY 11201, USA
shao.minghao@nyu.edu

ABSTRACT

In 1978, Morris described the very first streaming algorithm: the "Morris Counter" Morris (1978), which managed to provide a $(1+\epsilon)$-approximation counter with $1-\delta$ probability to the counter's value. Based on that, in 2022, Nelson et al.'s 2022 study significantly advanced this concept, optimizing space utilization and resolving the asymptotic space complexity issue in approximate counting in Nelson & Yu (2022). We propose Morris Decay, an improved algorithm with adaptive method on the ϵ value dynamically adjusting the bound to further reduce the error range particularly in high-count scenarios. Our research also extends previous developments, empirically evaluating the new algorithm's improvements over the Morris Counter in terms of space efficiency and error margin. We also completed a series of experiments identifying optimal parameter configurations that balance resource usage with algorithmic performance and evaluate them in realistic application scenarios.

1 INTRODUCTION

The genesis of the approximate counting algorithm can be traced back to the late 1970s with the introduction of Morris Counter Morris (1978) which marks a significant leap in efficient memory usage for counting tasks. Suppose one needs to count the total number of elements occurred in a continuous stream. One naive idea is to maintain a counter in the memory and adding one to the counter every time a new element coming in. In that case, the space consumption is $\mathrm{O}(\log_2 N)$, here we assume the total number of elements in that stream is N. Morris introduced a Monte Carlo-based "approximate counter" algorithm that allows for a constant factor approximation of a value N with high probability uses $(\mathrm{O}(\log_2 \log_2 N))$ memory bits Morris (1978). There are later works which analyzed in more detail on Morris Counter Flajolet & Nigel Martin (1985) and resulted that randomized space $\mathrm{O}(\log\log N + \log(1/\epsilon) + \log(1/\delta))$ is sufficient to return a $1+\epsilon$ approximation with success probability $1-\delta$; the space usage is variable and the given quantity represents its expected value. Additionally, this space requirement is likely to be met with high probability.

In Nelson & Yu (2022), Nelson et al gave an improved algorithm and a matching lower bound based on Morris Counter and its following analysis. Specifically, they demonstrated that the accurate relationship with the inverse failure probability is double logarithmic rather than single logarithmic. They proved that the failure probability with using $\mathrm{O}(\log\log N)$ is $1/\operatorname{poly}(\mathrm{N})$ instead of $1/\operatorname{poly}(\log \mathrm{N})$, mentioned in previous analysis. They also proved that for all Morris+ algorithms, including their new algorithms, any Morris Counter+ algorithm with $\log\log N + 2\log(1/\epsilon) + \log\log(1/\delta) + \mathrm{o}(1)$ bits of memory usage will output $\hat{N}$ as the estimation

result, which will satisfy $\mathbb{P}(|N - \tilde{N}| > \epsilon N) < \delta$ in high probability. They also demonstrated that by applying an important tweaking to Morris Counter, Morris Counter actually shares the same bound mentioned above, which can be regarded as an improved analysis of original Morris Counter. In refining the Morris Counter, a key modification involves the incorporation of deterministic counting at initial stages. This adjustment is critical for ensuring accuracy in the preliminary phases, as a single bit of deviation from expected results can significantly impact the final estimate. The accuracy of early counts is important; an error at this juncture can preclude the assurance of the counter's performance as a $(1 + \epsilon)$-approximation of N with a probability of δ. A mathematical justification for this modification has been provided to substantiate its necessity for maintaining the counter's efficacy in their paper.

In general, we do not expect that memory usage for stage one would exceed the final restriction of Morris Counter's maximum space usage, which is actually $\mathrm{O}(loglogN + log(1/\epsilon) + loglog(1/\delta))$. However, if for the deterministic counting stage, we set up its threshold as n, for that stage, our maximum space consumption is actually $\mathrm{O}(logn + log(1/\epsilon) + loglog(1/\delta))$, which should generally be considered less than $\mathrm{O}(loglogN + log(1/\epsilon) + loglog(1/\delta))$. This limits the range of setting constant C in the algorithm, which reveals a valuable problem to be discussed that how to set up the constant C to make this algorithm as accurate as possible with the prerequisite that we don't know the actual size of our stream. In this paper, we discussed some methodologies regarding selection of the constant C and its effect on the eventual estimation.

Also, despite the abundance of memory in contemporary computers, which vastly exceeds $\log N$ bits even for N approaching the total particle count in the universe, approximate counting remains a practical necessity. The real-world relevance lies not in single instances but in maintaining numerous counters simultaneously. In the paperNelson & Yu (2022), several real-world application scenarios such as "Least Frequently Used" (LFU) cache eviction policy in Redis multitude of counters and Morris' original intent, as he faced the task of tracking 2^{63} distinct counters for trigram occurrences in a spellchecker project were mentioned. However, for nowadays machine learning projects, larger and larger datasets are required, which places higher demands on the computer's memory. In that case, reducing memory usage per counter, even marginally, becomes significant.

Due to the proven result, the Morris Counter+ algorithm is a $1 + \epsilon$ approximation with probability δ. It guarantees that the error is in ϵ range on the same magnitude with N. If we gradually decay the ϵ with the increment of N by doing so, we are able to gradually narrow the product of ϵN, substantially reduces the margin of error, with the trade off that increase the space usage. Hence we provide 3 different strategies to decay the ϵ to narrow the error range with N increasing. Each of them focus on different aspects like space complexity and accuracy. And its' also feasible to increase ϵ with increment of N to save more spaces with large N.

An overview of the state-of-the-art approximate counting algorithms is provided in this section. These algorithms are invaluable irange of values of large n situations where traditional counting methods are impractical due to memory constraints because of their capacity to handle large amounts of data with little memory. These algorithms have evolved in tandem with streaming algorithms and have been impacted by advances in related fields such as computational complexity theory, data structures, and randomized algorithms.

1.1 MORRIS COUNTER

Robert Morris at Bell Laboratories first proposed the approximate counting algorithm known as the Morris Counter Morris (1978) in 1978. It uses as little memory as possible to estimate the number of elements in a stream by incrementing a counter through probabilistic techniques.

The Morris Counter's basic idea is to increment a counter X with a probability of $p < 1$ rather than incrementing deterministically with $p = 1$. Rather than storing the full count, which would require $\log N$ memory, the Morris Counter adopts a probability less than 1, so it only needs to store the counter X, which requires $\log \log N$ memory.

The original Morris Counter, in particular, increases X with a probability $p = 0.5^X$. This suggests that for each additional N elements in the stream, X should, on average, double. The total count

N can be approximated as $2^X - 1$ by keeping X constant. This estimator has a high variance that increases with N, despite being unbiased.

A closer look indicated that a can be tuned to reduce the variance when the increment probability is parameterized as $p = (1 + a)^{-X}$. With this modification, the Morris Counter still only requires $\log \log N + O(1)$ memory, but can provide accuracy guarantees like $|N - \hat{N}| \leq \varepsilon N$ with a probability $\geq 1 - \delta$.

The ability of the Morris Counter to provide approximate counts using sub-logarithmic memory is its main benefit. Because of this feature, it can be used to count problems where memory is very limited but approximate counts are sufficient. Modern systems with limited memory resources, such as network routers, still use variations of the Morris Counter.

1.2 Other Approximation Methods

In Yun & Liu (2020), new techniques such as the general Morris counters were examined for their statistical properties and variance after the Morris Counter. Kruskal & Greenberg (1991) used a tailored function f and probability sequence $P(c)$ to achieve enhanced approximate counting. Albers & Mitzenmacher (1997) presented a lower bound on the competitive ratio of counting algorithms. Morris's 1978 algorithm was enhanced by Cvetkovski (2007) using Adaptive SLAC and Sampled Log Approximate Counting (SLAC), which optimized memory access and reduced error in 99

Gronemeier & Sauerhoff (2009) combined continuous approximate counting and approximate reservoir sampling for sublogarithmic space counting in data streams, resulting in space complexity with a logarithmic dependence on stream length n.

Using martingales, Rosenkrantz (1987) examined approximate counting algorithms, emphasizing the counter's capacity to track actual counts and proving its efficacy through moments, unbiasedness, and concentration.

1.3 Morris+ Accuracy Bound

This work builds upon Nelson & Yu (2022), which suggested a better algorithm by analyzing the bound of the Morris Counter. Morris Counter states that, given a universal constant $C' > 0$, there is a lower bound on accuracy that satisfies the following formula:

$$P[|\hat{N} - N| > C'\epsilon N] < C'\delta$$

where $\hat{N}$ is the approximation of the counting number produced by the algorithm, and N is the actual number of new elements. Similar to the original Morris Counter, this algorithm uses two hyperparameters: ϵ, which represents the algorithm's acceptable error and a larger value would allow for a larger difference between the estimated output and the actual element number, and $\delta \in (0, 1/2)$, which represents the probability that the algorithm will approximate the total number of elements in the stream by $1 + \epsilon$.

1.4 Morris+ Memory Bound

In the Nelson & Yu (2022) paper, their work includes a proof of their algorithm's space usage's upper bound with high probability. To be specific, their algorithm is analyzed for its space complexity concerning the storage of two variables X and Y, which are incremented probabilistically. The main theorem posits that the probability the algorithm needs more than $\log \log N + \log \log(1/\delta) + 3 \log(1/\epsilon) + \Omega(t)$ bits of memory after N increments is bounded above by $(\epsilon/N)^{2^t}$, for any $t \geq C \cdot (\log \log N + \log \log(1/\epsilon))$, where C is a sufficiently large constant. The proof exploits the logarithmic storage of variable X and the constant-factor space overhead due to the cubic dependence of α_{new} on $1/\epsilon$ in the algorithm's complexity.

2 Approach

In Nelson & Yu (2022), their algorithm requires a large constant C in order to obtain the promise of space consumption will not exceed its upper bound $\log \log N + \log \log(1/\delta) + 3 \log(1/\epsilon) + \Omega(t)$ bits

of memory with probability at most $(\epsilon/N)^{2^t}$. However, the largest constant C also plays an important role on deciding the stage transition threshold. In the algorithm, $X_0 = \lceil ln_{1+\epsilon}(Cln(1/\eta)/\epsilon^2)$, whereas when the actual X is less than X_0, the algorithm will remain on stage one, following the deterministic counting method. This deterministic counting stage is crucial in Morris+ algorithm as it ensures the probability of giving an $1+\epsilon$ approximate estimation. However, this implies that if one overestimates the upcoming stream's scale, it is possible set a large constant enabling the whole algorithm to remain on the deterministic counting stage. In this case, the whole algorithm will degrade as a deterministic counting algorithm, which will consume too much space, definitely not what we expect. In that case, setting up the constant C becomes a crucial problem to Morris+.

To resolve this problem, we set up a series of experiments to explore a feasible strategy of selecting and giving an upper bound for this constant C, restricting the algorithm's performance within error while also guaranteeing the algorithm would not degrade as a purely deterministic counter which takes $\mathrm{O}(\log N)$ bits of memory spaces instead of what is given in the paper Nelson & Yu (2022).

The error provided in Morris+ algorithm is ϵ, which promises this algorithm will provide an estimation with error range in $(-\epsilon, \epsilon)$ with probability $1-\delta$. But when the number of elements in a stream is massive, ϵN will still have a large absolute error compared with deterministic counting. In this section, we explore the method of asymptotically reduce the parameter ϵ, which will potentially provide a narrower error range with trading off more space consumption.

2.1 A LARGE ENOUGH C

For Morris+, it is crucial to have its deterministic counting stage, which can be regarded as a normal traditional counter enables accurate results on counting small numbers. However, due to this counting method's feature, for this stage, the space complexity is $\mathrm{O}(\log N)$. If one wants to obtain a tight bound on space consumption using Morris+ algorithm, it is reasonable to assume that the maximum space consumption in this task should not exceed the final space consumption $\mathrm{O}(\log\log N + 3\log(1/\epsilon) + \log\log(\delta))$. With this restriction, for the deterministic counting stage, when doing the deterministic counting, the space consumption is hence determined. Let n denote the increment of elements in the stream for the deterministic counting stage; for Morris+ to transfer from deterministic counting to approximate counting, we have:

$$\log n \leq \log\log N \tag{1}$$

Where N is the total number of elements in the stream.

Notice that in Morris+, the stage transfer happens when $Y > \alpha T$, and before this transfer, $\alpha = 1$, which means Y will keep increasing with probability 1, until it first satisfies $Y > \alpha T$, at which time Morris+ will switch to approximate counting.

$$Y = \lceil (1+\epsilon)^{X_0} \rceil \tag{2}$$

Thus we have our upper bound for Y:

$$\log Y \leq \log\log N \tag{3}$$

$$\log(\lceil (1+\epsilon)^{X_0} \rceil) \leq \log\log N \tag{4}$$

$$\lceil \frac{C\ln(1/\eta)}{\epsilon^3} \rceil \leq \log N \tag{5}$$

$$C \leq \frac{\epsilon^3 \log N}{\ln(1/\eta)} = \frac{\epsilon^3 \log N}{\ln(1/\delta)} \tag{6}$$

With C not exceeding $\frac{\epsilon^3 \log N}{\ln(1/\delta)}$, there is a guarantee that the space consumption for all stages will not exceed the upper limit of $\mathrm{O}(\log\log N + \log(1/\epsilon) + \log\log(1/\delta))$.

In the next section, we display the results of running several rounds of simulation based on this prerequisite with different combinations of parameters ϵ and δ, proving that with this limit, the constant C is still "large enough" to give an $1+\epsilon$ approximate estimation to N.

2.2 ADAPTIVE BASE OF MORRIS+

With the given proof of Morris+, it is promised that Morris+ is a $1+\epsilon$ approximate estimation. To have a tighter error range, smaller ϵ can be selected, which leads to the increasing space consumption. In the paper Nelson & Yu (2022), it is claimed that the upper bound for their Morris+ algorithm is $\mathrm{O}(\log\log N + \log(1/\epsilon) + \log(1/\delta))$, but in fact with careful examination about this upper bound, we find the real time complexity is determined by different combinations of ϵ and δ, which can be represented as $\mathrm{O}(\log_2 \log_{1+\epsilon} N + \log_2(1/\epsilon) + \log_2(1/\delta))$. In that case, a smaller ϵ leads to a larger space of memory. When it comes to estimate a pretty sizeable stream, we have the absolute deviation for our estimation equals to $|N - \hat{N}| < \epsilon N$ with probability $1-\delta$. With a larger ϵ, comes withe a larger absolute error. To get a more accurate estimation, one can choose to use a smaller ϵ to obtain a tighter error range, however, this will largely increase the space consumption at the same time.

Without knowing the exact scale of the upcoming stream, if one wants to obtain a tight error range, it is required to set up a small ϵ. However, when the size of the stream does not match the size estimated in advance (as indeed happens frequently in real-world scenarios), people may waste unnecessary space on maintaining the relatively small absolute error (expecting the upcoming stream's size to be larger) with setting a small ϵ on a relatively smaller stream. To avoid this, we give 3 strategies which allow Morris+ to adjust the base of $(1+\epsilon)^X$, which ensure larger ϵ is matched with larger stream while smaller ϵ is matched with smaller scale of data. We can have better accuracy when the stream is smaller while also enabling better space complexity when the stream is larger.

In experiments, we offer three potential decay strategies on updating ϵ, with different characteristics, leading to different effects in different application scenarios.

Define t as the number of epochs in the approximate counting stage, i.e. t denotes the increment of X in the second stage in Morris+, which begins for the first time when $Y > \alpha T$ and $t = X - X_0$.

2.2.1 POLYNOMIALLY UPDATING ϵ

For the first strategy, we update the ϵ polynomially; with the increment of t, ϵ will be deducted with a fix value λ, which defines the decreasing rate and can be adjusted in advance as a parameter. Hence we have:

$$\epsilon = \epsilon_0 - \lambda t$$

2.2.2 EXPONENTIALLY UPDATING ϵ

For the second strategy, we update ϵ exponentially, which leads to a stronger substantial reductions with the increment of t. In that case, for larger stream, the absolute error can be guaranteed within in a smaller range which has a stronger accuracy with more space consumption. The update strategy can be described as:

$$\epsilon = \epsilon_0 e^{-\lambda t}$$

2.2.3 LOGARITHMICALLY UPDATING ϵ

For the last strategy, we update ϵ logarithmically, which results in a relatively smoother reduction on ϵ. By doing so, with larger stream, the ϵ will decrease at a progressively lower rate, which offers a good promise on space consumption. This strategy is helpful for memory usage sensitive scenarios, making sure larger stream and probably expectation exceeding elements will not consume too much spaces. This strategy can be expressed as:

$$\epsilon = \epsilon_0 \frac{\lambda}{\ln t} (t \geq 3)$$

Algorithm 1 Polynomial Decay

procedure POLYNOMIAL DECAY(ϵ, δ)
    Init():
    $\eta \leftarrow \delta, X_0 \leftarrow \left[1 + \eta \left(C \ln(1/\eta)/\epsilon^3\right)\right]$
    $Y \leftarrow 0, X \leftarrow X_0, \alpha \leftarrow 1, T \leftarrow [(1+\epsilon)X]$

    Increment():
    with probability α, **update** $Y \leftarrow Y + 1$
    if $Y > \alpha T$ **then**
        $T \leftarrow [(1+\epsilon)^X], \eta \leftarrow \frac{\delta}{X^2}$
        $X \leftarrow X + 1$
        $\epsilon \leftarrow \epsilon_0 - \lambda t$
        $\alpha_{\text{new}} \leftarrow \frac{C \ln(1/\eta)}{\epsilon^3 T}$
        $Y \leftarrow \left[Y \cdot \frac{\alpha_{\text{new}}}{\alpha}\right]$
        $\alpha \leftarrow \alpha_{\text{new}}$
    end if

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

Algorithm 2 Exponential Decay

procedure EXPONENTIAL DECAY(ϵ, δ)
    Init():
    $\eta \leftarrow \delta, X_0 \leftarrow \left[1 + \eta \left(C \ln(1/\eta)/\epsilon^3\right)\right]$
    $Y \leftarrow 0, X \leftarrow X_0, \alpha \leftarrow 1, T \leftarrow [(1+\epsilon)X]$

    Increment():
    with probability α, **update** $Y \leftarrow Y + 1$
    if $Y > \alpha T$ **then**
        $T \leftarrow [(1+\epsilon)^X], \eta \leftarrow \frac{\delta}{X^2}$
        $X \leftarrow X + 1$
        $\epsilon \leftarrow \epsilon_0 e^{-\lambda t}$
        $\alpha_{\text{new}} \leftarrow \frac{C \ln(1/\eta)}{\epsilon^3 T}$
        $Y \leftarrow \left[Y \cdot \frac{\alpha_{\text{new}}}{\alpha}\right]$
        $\alpha \leftarrow \alpha_{\text{new}}$
    end if

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

Algorithm 3 Logarithmic Decay

procedure LOGARITHMIC DECAY(ϵ, δ)
  Init():
  $\eta \leftarrow \delta, X_0 \leftarrow \left[1 + \eta\left(C\ln(1/\eta)/\epsilon^3\right)\right]$
  $Y \leftarrow 0, X \leftarrow X_0, \alpha \leftarrow 1, T \leftarrow [(1+\epsilon)X]$

  Increment():
  with probability α, **update** $Y \leftarrow Y + 1$
  if $Y > \alpha T$ **then**
    $T \leftarrow [(1+\epsilon)^X], \eta \leftarrow \frac{\delta}{X^2}$
    $X \leftarrow X + 1$
    if $t \geq 3$ **then**
    $\epsilon \leftarrow \epsilon_0 \frac{\lambda}{\ln t}$
  end if
  $\alpha_{\text{new}} \leftarrow \frac{C\ln(1/\eta)}{\epsilon^3 T}$
  $Y \leftarrow \left[Y \cdot \frac{\alpha_{\text{new}}}{\alpha}\right]$
  $\alpha \leftarrow \alpha_{\text{new}}$

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

2.2.4 ANALYSIS OF THESE 3 STRATEGIES

Notice that each λ in above strategies has various initial value and meaning, needed to be set up individually according to which strategy is chosen.

Even though for above 3 strategies, we have 3 different methods to update ϵ, the expectations of those strategies follow the same expressions. Let's denote ω_k as the number of increments required for X to advance epoch k to epoch $k+1$, Y_k denote the first time in this epoch $Y > \alpha T$, which can be expressed as $Y_k = \lfloor \alpha_k T_K \rfloor + 1$ we have:

$$\mathbb{E}(\omega_k) = (Y_k - Y_{k-1})/\alpha \tag{7}$$
$$= T_k - T_{k-1} \tag{8}$$

And hence our gross expectation of all elements in the stream is:

$$\mathbb{E}(\hat{N}) = \sum_{i=0}^{k} \mathbb{E}(\omega_k) \tag{9}$$
$$= T_k - T_{k-1} + T_{k-1} - T_{k-2} + .. + T_1 - T_0 \tag{10}$$
$$= T_k - T_0 \tag{11}$$
$$= (1+\epsilon)^X \tag{12}$$

Hence we can roughly prove that T_k is the correct expectations for all 3 strategies. Thus $T_K = \hat{N}$, which can be returned as a result for each Query(). And with adapting ϵ, no more spaces are needed which is crucial for an algorithm focusing on saving spaces of memroy.

With these 3 different strategies, Morris+ algorithms can be tuned to focus on different aspects, with the third strategy being chosen for space-consumption-sensitive counters and the second for accuracy-oriented applications. On these 3 strategies are set to solving problems caused by unexpected stream size. For applications that are extremely sensitive to space consumption, ϵ can even be set to gradually increase to handle unexpected large stream which will cause massive space consumption.

2.3 Comparison with Morris Counter

To compare the original Morris Counter with Morris+, we compare several measures of model performance. Using a set δ and ϵ value, such as $\delta = 0.05, 0.25$ and $\epsilon = 0.05, 0.25$, the two counters are run with a target of $N = 10^7$, and compared with the ϵN theoretical bound as well as the naive count. The value of C used was $C = \frac{\log_2(N)\cdot\epsilon^3}{-\ln(\delta)}$, dependent on the max number of increments N. The corresponding α in the original Morris counter was set to $\alpha = 2\epsilon^2\delta$, as described in Nelson & Yu (2022) in order to give a similar theoretical bound as Morris+, see 1.3.

3 Experiment and Results

An overview of the experiment in this study on the assessment and analysis of our strategy, which was suggested in Section 2, is given in this section. Initially, we ran the experiment on the Morris+ bound, providing an approximate target of N = 10,000,000. Secondly, we studied Morris+'s space complexity by putting all of the counters and variables into binary format. We also conducted a thorough comparison of the counting increment and variance between Morris+ and the original Morris Counter Morris (1978).

To provide the reader with a fundamental understanding of the algorithm's performance based on varying precision, we tested the algorithm using various ϵ and δ configurations of algorithmNelson & Yu (2022). Parameters were adjusted within a range: δ and ϵ were both adjusted within the range 0.05 to 0.5, and C from 0.05 to 0.5 (with N number of increments) both with step size 0.05. The number of increments, N was set to 10000000 trials. We run the experiment for each experiment configuration 10 times in order to mitigate the negative effects of Morris Counter and Morris+'s randomness. Then, we remove the maximum and minimum values from the approximation over the ten runs, and take the average.

The objective of the experiment was to experimentally confirm the relationship between parameters and the observed performance of the algorithm, including various trade-offs such as space complexity and accuracy.

3.1 Evaluation of Morris+

Morris+ was assessed using two metrics: bound and space complexity. By converting the algorithm's variables to binary and estimating the number of bits each related variable occupied on the register, we were able to estimate the space occupation from the simulation in the space complexity section.

We gave the algorithm bound evaluation on two bases: the error, which is the difference using the formula

$$E_{\epsilon,\delta,N} = \frac{|\hat{N} - N|}{N}$$

between the estimated approximation $\hat{N}$ and the actual number of elements N. where the hyperparameters in the experiment configuration are ϵ and δ. Additionally, we gave the Morris+'s absolute bound in comparison to our experiment, $N = 10,000,000$. In order to ascertain the lower bound of bits needed for the register, we recorded the variable value for each epoch in the algorithm iteration and, for each configuration, we took the maximum value of the entire experiment epoch.

3.1.1 Space Complexity

Figure 1 takes into account the two algorithmic counters, X and Y, and eliminates all other variables. With only the counters X and Y in the Morris+ taken into account, Figure 2's space complexity ranges from 12 to 18, suggesting that, for a number $N = 10,000,000$, a register with a total of 18 bits should be sufficient to hold all the variables. It suggests that, in the worst scenario, a register with 18 bits would be adequate to count a large number of $10,000,000$ for larger δ and smaller ϵ, while in the best scenario, with $\delta = 0.05$ and $\epsilon = 0.5$, 12 bits would be sufficient.

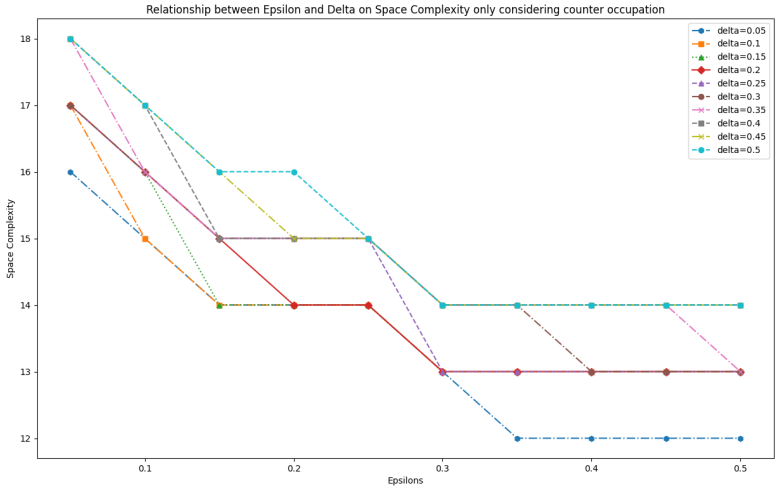

Figure 1: Space complexity only considering counter X and Y with different configurations of ϵ and δ

3.1.2 Sufficient Large C

Figure 2 consisted of different combinations of ϵ and δ. For each combination, their constant C is defined as $\frac{\epsilon^3 \log N}{\ln(1/\delta)}$ in 3. Those results of all the combinations managed to fall within their respective defined error intervals with corresponding constant C, which suggests that our hypothesis about constant C holds with high confidence.

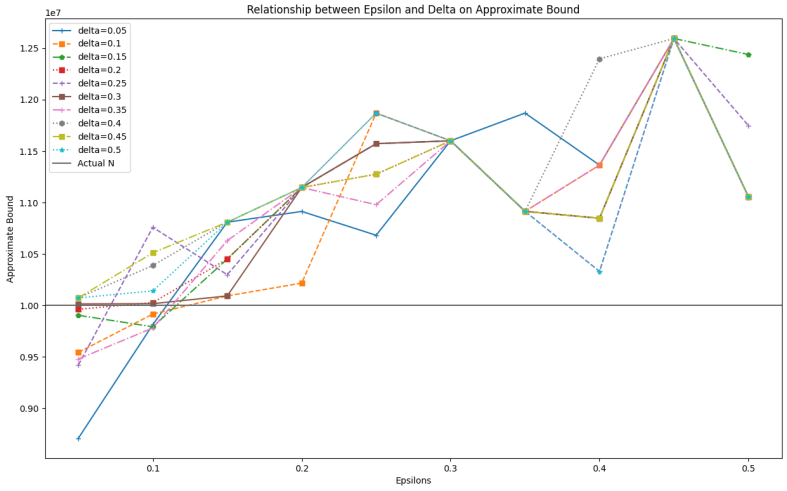

Figure 2: Deviations of different combinations of ϵ and δ with their corresponding C

3.2 Epsilon Decay

The general approximation boundary for the three decay strategies we suggested is shown in Figure 5. The data reveals a complex relationship between ϵ, δ, and decay type; no single decay method

consistently outperforms the others at all ϵ and δ values with around 10%; however, in the majority of the ϵ and δ configurations, the exponential decay outperforms the others. Certain decay types appear to minimize error more effectively at particular ranges of ϵ for some δ values. Interestingly, there is a significant divergence in error rates at higher ϵ values, indicating that the decay type selection becomes crucial in this range.

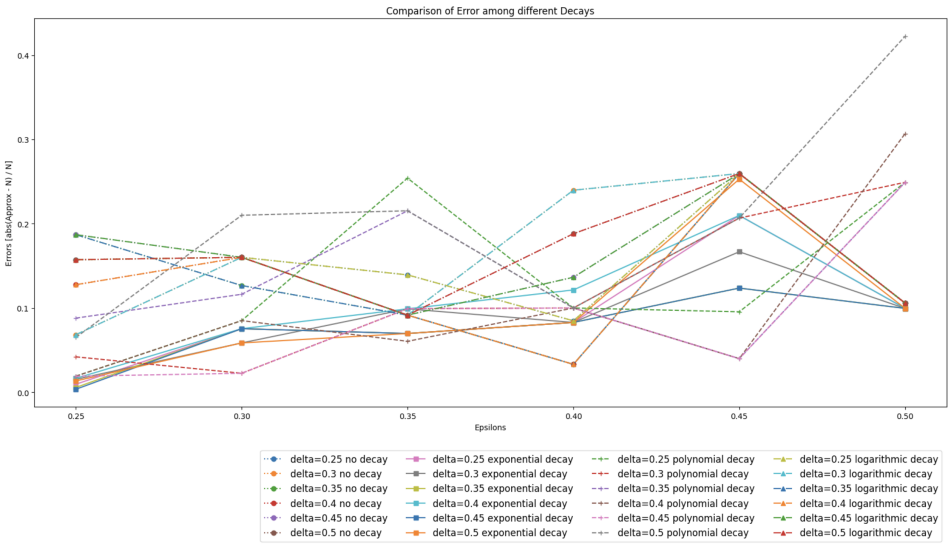

Figure 3: Bound difference comparison of approximation of $\hat{N}$ and actual counting number N for all three decay strategies, $\lambda = 0.0001$ for exponential decay, $\lambda = 0.001$ for exponential decay,$\lambda = 1$ for logarithmic decay

Figure 4 displays all space consumption situations of all 3 decay strategies and the original Morris+. This aspect explicitly demonstrate the space complexity features of these decay strategies. In general, exponential decay strategy with more rapidly decreasing ϵ uses more memory space trading off a relatively high accuracy while logarithmic decay has a slower decreasing rate which is potentially more space-saving.

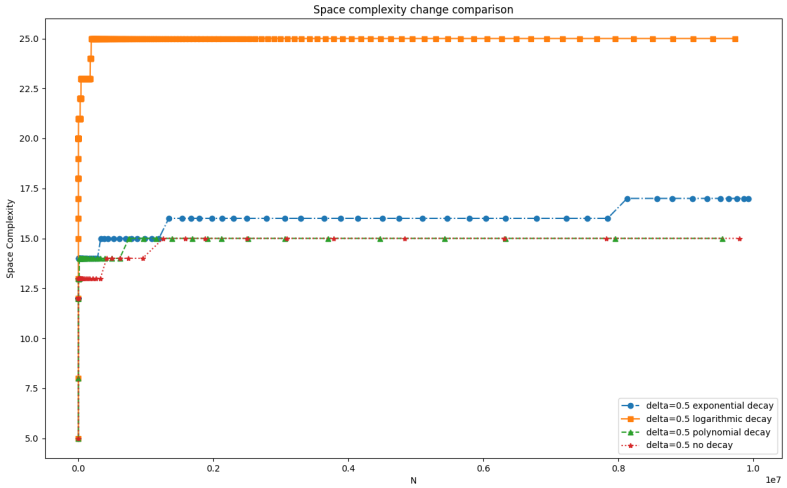

Figure 4: Space consumption of 3 different decay strategies and original Morris+

3.2.1 ϵ OVER EPOCH ON THREE DECAY STRATEGIES

Figures 5, 6, and 7 show the change of ϵ over epoch for all three decay strategies with setting the initial $\epsilon = 0.25$ and gradually decreases as the epoch increases to 80. The exponential and consistent nature of the decay suggests a controlled reduction, likely to fine-tune the process's performance over time.

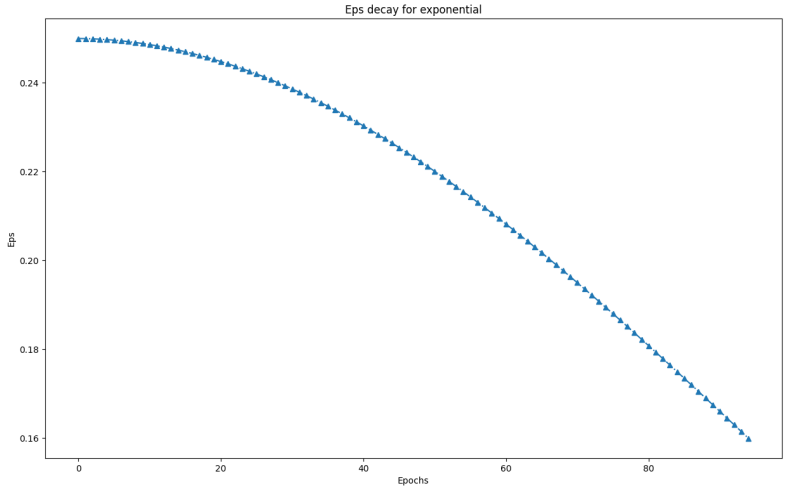

Figure 5: Change of ϵ for exponential decay over epoch

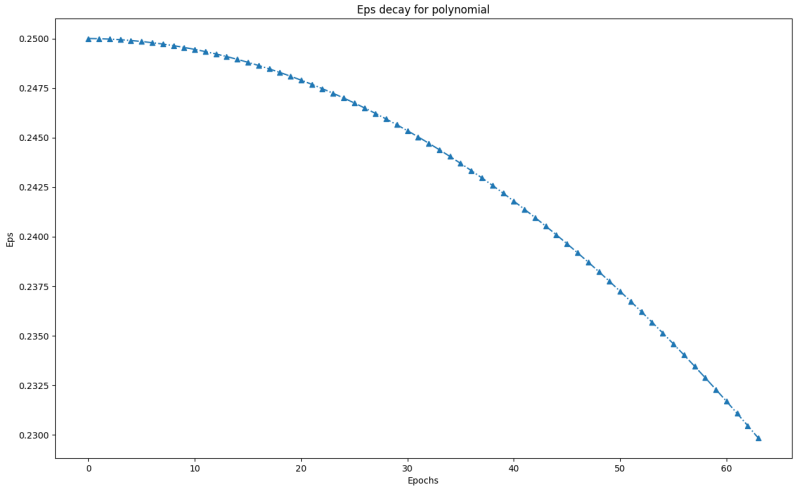

Figure 6: Change of ϵ for polynomial decay over epoch

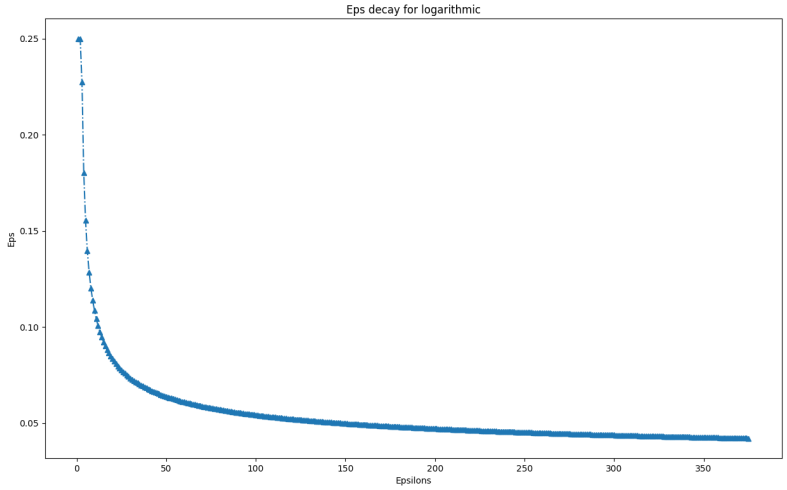

Figure 7: Change of ϵ for logarithmic decay over epoch

3.3 Comparison between Morris Counter and Morris+

The original Morris counter exhibits drift due to the randomness of the algorithm, which can increasingly render the counter inaccurate over time. The Morris+ algorithm, although can have a larger margin of error from the actual count than the original Morris counter, is usually more reliable as it tends to stay within the ϵN bound most of the time (a guarantee which is not certain for the original Morris counter). Thus, the Morris+ counter gives a tuneable estimate (bound) of its accuracy depending on the setting of its hyper-parameters.

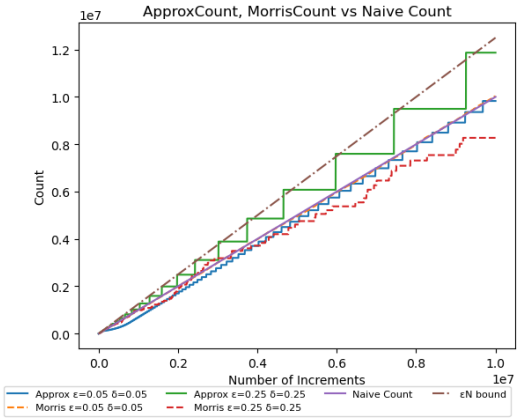

Figure 8: ApproxCount (Morris+) vs. Morris Counter

4 Conclusion

In this study, we have addressed several crucial aspects of the Morris+ algorithm and its performance, focusing on the determination of the constant C and the adaptability of the algorithm to varying stream sizes and error requirements. Our findings provide valuable insights into the practical implementation and optimization of Morris+ for different applications.

First, we explored the role of the constant C in the Morris+ algorithm. We observed that while a large value of C is necessary to ensure the space consumption remains within the specified upper bound, it also influences the stage transition threshold. This deterministic counting stage is pivotal in Morris+ as it guarantees the probability of providing a $1 + \epsilon$ approximate estimation. However, an excessively large constant C can lead to Morris+ behaving like a deterministic counting algorithm, which consumes more space than desired. To address this issue, with prerequisites that entire algorithm does not consume more space at any stage than the premise of $\mathrm{O}(\log\log N + \log(1/\epsilon) + \log(1/\delta))$, we conducted experiment to test if our given C is still large enough to ensure both error control and prevents excessive space consumption.

Our analysis demonstrated that by setting C not to exceed $\frac{\log_2 N\epsilon^3}{\ln(1/\delta)}$, we can restrict the space consumption of all stages within the desired limit of $\mathrm{O}(\log\log N + \log(1/\epsilon) + \log\log(1/\delta))$. This ensures that Morris+ strikes a balance between error control and memory usage, making it a practical choice for various applications.

Furthermore, we explored the adaptability of Morris+ by introducing strategies for adjusting the base $(1 + \epsilon)^X$ based on the estimated stream size. These strategies enable Morris+ to dynamically tailor its error tolerance to the scale of the incoming stream, preventing unnecessary space consumption when the stream size estimation is inaccurate.

In the pursuit of tighter error bounds, we also examined the trade-off between error and space consumption in Morris+. Smaller values of ϵ can provide narrower error ranges but result in increased memory usage. Our analysis highlighted the importance of choosing an appropriate ϵ value based on the expected stream size to strike a balance between accuracy and memory efficiency.

Finally, we compared Morris+ with the original Morris Counter, evaluating its performance in terms of error control and memory consumption. Our results demonstrated that Morris+ offers a competitive alternative, providing accurate estimations while ensuring memory usage remains within acceptable bounds.

In conclusion, the insights and strategies presented in this paper contribute to a better understanding of Morris+ and its practical implementation. By carefully selecting the constant C, adjusting the base $(1 + \epsilon)^X$, and considering the trade-offs between error and memory usage, Morris+ can be tailored to suit various real-world scenarios, making it a valuable tool for approximate counting applications.

5 Future Work

We provide three strategies to gradually adjust ϵ. They are not just feasible on decreasing ϵ but are also valuable for increasing ϵ epoch by epoch. In this case, increasing ϵ will make the algorithm more space-efficient with larger N, which can save space for handling unexpectedly large scale streams. Increasing ϵ is more in line with Morris counter's space-saving philosophy.

This paper focused on the experimental verification of the Morris decay algorithm. Although we provide a simple proof on why Morris decay algorithm is a correct estimation based on Morris+, more rigorous mathematical analysis is needed to analyze this algorithm in detail.

References

Susanne Albers and Michael Mitzenmacher. Revisiting the counter algorithms for list update. *Information processing letters*, 64(3):155–160, 1997.

Andrej Cvetkovski. An algorithm for approximate counting using limited memory resources. *ACM SIGMETRICS Performance Evaluation Review*, 35(1):181–190, 2007.

Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, October 1985. ISSN 00220000. doi: 10.1016/0022-0000(85)90041-8. URL https://linkinghub.elsevier.com/retrieve/pii/0022000085900418.

André Gronemeier and Martin Sauerhoff. Applying approximate counting for computing the frequency moments of long data streams. *Theory of Computing Systems*, 44:332–348, 2009.

Joseph B Kruskal and Albert G. Greenberg. A flexible way of counting large numbers approximately in small registers. *Algorithmica*, 6:590–596, 1991.

Robert Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, October 1978. ISSN 0001-0782, 1557-7317. doi: 10.1145/359619.359627. URL https://dl.acm.org/doi/10.1145/359619.359627.

Jelani Nelson and Huacheng Yu. Optimal Bounds for Approximate Counting. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pp. 119–127, Philadelphia PA USA, June 2022. ACM. ISBN 978-1-4503-9260-0. doi: 10.1145/3517804.3526225. URL https://dl.acm.org/doi/10.1145/3517804.3526225.

Walter A. Rosenkrantz. Approximate counting:a martingale approach. *Stochastics*, 20(2):111–120, February 1987. ISSN 0090-9491. doi: 10.1080/17442508708833439. URL http://www.tandfonline.com/doi/abs/10.1080/17442508708833439.

Tong Yun and Bin Liu. A deep analysis on general approximate counters. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp. 1231–1240. IEEE, 2020.

A Appendix

Additionally, we experimented with the ϵ, a modified version of our exponential decay function, which can be written as follows:

$$\epsilon = \epsilon_0 e^{\lambda t}$$

The space complexity for the elevated exponential ϵ was ascertained.

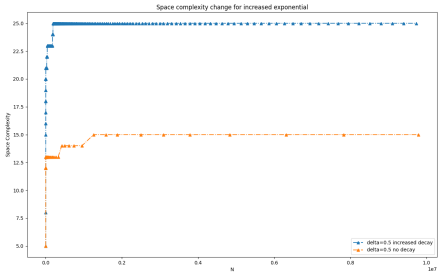

Figure 9: Space complexity of approximation of $\hat{N}$ and actual counting number N for increased exponential ϵ with $\lambda = 0.0001$

Figure 9 illustrates how the space complexity rises with increasing ϵ in comparison to the original Morris+.

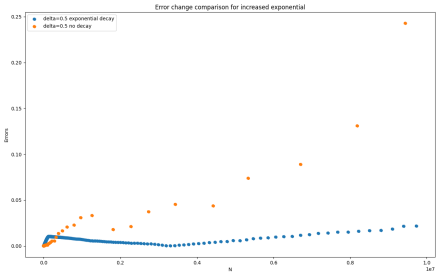

Figure 10: Space complexity of approximation of $\hat{N}$ and actual counting number N for increased exponential ϵ with $\lambda = 0.0001$

A tighter bound than the original algorithm is shown in Figure 10, which compares the bounds of the increased exponential ϵ and the original Morris+.