Notes on Alpha Algorithm

three authors

ABSTRACT

no abstract yet

1. ATTEMPTED OVERVIEW

We are the designers/builders/users of a sketch-based Big Data system. We were motivated by limitations of all existing sketches to design a new kind of sketch. We believe that this new sketch type hits a sweet spot of speed, accuracy, and simplicity.

A sketch must have numerous good features to be useful for us. This long list includes many things that everyone wants and are therefore always discussed in papers, such as good accuracy and low resource usage. However, we have some specific requirements that have been much less discussed in the literature, and which strongly drove our design process. These requirements are centered around the topic of "set expression evaluation".

1. We want accurate answers for set expressions. Therefore:
    (a) We do not use HLL. It is known that much better support for set operations (i.e. more operations supported, and better accuracy achieved) is possible by using a sketch type that include a set of hashes that can be used to estimate the selectivity of a set expression.
    (b) While certain commonly used rules for selectivity estimation discard hashes that in fact could be retained, our rule includes those samples in the calculation, thus achieving better accuracy.
2. We want flexible evaluation of set expressions. Therefore:
    (a) We made sure that our rule for set expression evaluation yields the same result even when the set expression is re-arranged to a logically equivalent form. (This is not true for many existing rules).
    (b) We made sure that our sketch data structure is powerful enough to represent, with no loss of accuracy, an intermediate result in the evaluation of a larger set expression.

All four of these points were achieved via a new sketch type, that is, a data structure with associated rules and algorithms, *all* of which are extremely simple. The data structure contains just three things:

1. A user-specified parameter k, which is a *soft* target for the size of the sketch.
2. An internal variable θ, which is a threshold against which hashes can be compared.
3. A set S of hashes, specifically the set of all hashes in the stream (so far) which are less than θ.

The size estimation rule for a sketch (k, θ, S) is simply $|S|/\theta$.

The sketch representing the output of the binary set operation Δ applied to sketches (k_1, θ_1, S_1) and (k_2, θ_2, S_2) is simply $(\infty, \theta_3 = \min(\theta_1, \theta_2), \{(h < \theta_3) \in (S_1 \Delta S_2)\})$.

Both of those rules are straightforward consequences of the relationship between θ and S.

So far, we haven't really used k. Its function is to limit the amount of offline storage used by the system's base sketches, that is, the sketches constructed directly from streams.[1]

Interestly, the fact that k is a *soft* target unlocks a sketch construction algorithm (see Figure ??) that is remarkably fast and simple. Compared to the obvious heap-based implementation for constructing a KMV sketch, this algorithm uses less than $2/3$ as much memory, and its update time is faster by a factor of $\log k$ in theory, and a factor of 3 in practice when $k = 2^{16}$.

2. A NEW ALGORITHM FOR BUILDING KMV-LIKE SKETCHES

In this section we develop an algorithm that is faster and uses less space than a heap-based implementation of KMV (as suggested e.g. by the Beyer Paper), but only approximately hits the size target for the sketch. However, we will show that the actual size of the sketch is tightly concentrated around the size target, and the error bounds on the resulting estimates are similar to those of KMV.

Sketch Construction Algorithm	Sketch Update Cost	Size in Memory
Heap-based	$O(\log k)$	$c_h \cdot k + k$
This paper	$O(1)$	$c_h \cdot k$

In the above table, N is the number of items in the stream, U is the number of unique items in the stream, k is the "target" size of the sketch. c is a positive constant bounded away from 0, that for concreteness can be thought of as being 1.

[1]Base sketches are the predominant consumers of offline storage in our system. "Derived Sketches" occupy less storage, and if we tried to limit their size, we would sacrifice the two "flexible evaluation" properties that were mentioned above.

2.1 New Sketch Construction Algorithm

2.1.1 Background and Motivation

The basic algorithm for constructing a KMV sketch uses a priority queue to keep track of the current k'th minimum hash value, and a dictionary to detect duplicate hashes. This algorithm *could* be implemented using a single data structure that supports both types of operations, such as a red-black tree or one of the Giroire-like ordered dictionary schemes from the algorithms literature. However, red-black trees are known to be slow in practice, and the above-mentioned ordered dictionary schemes are complicated and hence rarely used.

However, there is a simple way to achieve low update times: simultaneously use a heap for the priority queue and a hash table for the dictionary [cite Beyer]. Assuming that hash values occupy a single word of memory, then this approach uses k words for the heap, and $c_h \cdot k$ words for the hash table, where the constant $c_h > 1$ provides the headroom that the hash table requires. The sketch update time is dominated by the cost of updating the heap, namely $O(\log k)$.

The idea for a new and more efficient algorithm for building KMV-like sketches can be derived via the following intuition. Because it seems wasteful to be storing the same hash values in two different data structures, it would be nice to get rid of one of them, especially the priority queue, given its $O(\log k)$ update cost. However, eliminating the priority queue would sacrifice the ability to efficiently keep track of the exact current value of m_k (the k'th minimum value). One might hope that there exists a correct algorithm that can get by with a cheaply computable approximation of the current value of m_k. A plausible candidate for such an approximation is $(\frac{k}{k+1})^i$, where i is the number of times that a new hash value has been inserted into the sketch.

In the rest of this section we will describe and prove the correctness of an efficient new algorithm based on the above intuitions. If the dictionary is implemented using a hash table, then the space requirement of the new algorithm is $c_h \cdot k$ words, and its amortized update cost is $O(1)$. We will prove that the number of samples in the resulting sketch has expected value k, and variance less than $\frac{k}{2} + \frac{1}{4}$.

Interestingly, we designed this algorithm primarily for speed, and were anticipating that the newly introduced variability in $|S|$ would cause its estimates to have higher variance than those of KMV. However, when the new estimates were analyzed, they turned out to have *lower* variance than those of KMV, even when KMV is given one extra sample. See section ??.

[We will also present numerical evidence, based on an exact error analysis evaluated using exact arithmetic, that the estimates produced by this sketch algorithm have expected value n, and variance $((2k+1)n^2 - (2k^2+2k+1)n + (k^2+k))/(2k^2)$, which is approximately $n^2/(k-\frac{1}{2})$ for $n \gg k \gg 1$. This variance is less than that of a KMV sketch with same target size k, or even KMV with target size $k+1$.]

Finally, we will present empirical results that are very similar to the above theoretical calculations. This similarity implies that the production-oriented hash functions that we employ (namely City Hash and Murmur Hash) aren't sufficiently different from true random values to throw off either our error analysis or our running time analysis.

2.2 The New Algorithm

The new algorithm works by keeping track of the set S of all hash values ever seen that are less than a threshold θ. This threshold is gradually reduced using an updating policy that causes the size of S to be concentrated around a target size k. Pseudocode is presented as Algorithm ??. The algorithm's parameters are a stream $\mathcal{H}$ of hashes, a threshold updating factor α, and an initial set size w. These parameters were chosen to facilitate analysis. In practice the natural parameters are the stream $\mathcal{H}$ and the target size k; the code as presented would be called with $w = k$ and $\alpha = k/(k+1)$.

Algorithm 1 Basic Algorithm $(\mathcal{H}, \alpha, w)$

```
1: $S \leftarrow$ (First $w$ unique hashes in $\mathcal{H}$).
2: $\theta \leftarrow 1.0$
3: **Loop:**
4: {Invariant: $E(|S|) = \alpha/(1-\alpha)$}.
5: {Invariant: S contains all $h < \theta$ seen so far in $\mathcal{H}$}.
6: **if** no more hashes in $\mathcal{H}$, **return** $(\theta, |S|)$.
7: $h \leftarrow$ next hash in $\mathcal{H}$.
8: **if** $h \geq \theta$, **goto Loop**.
9: **if** $h \in S$, **goto Loop**.
10: $S \leftarrow S \cup \{h\}$.
11: $\theta \leftarrow \alpha \cdot \theta$.
12: $S \leftarrow S \setminus \{(x \geq \theta) \in S\}$.
13: **goto Loop.**
```

2.3 Analysis

2.3.1 Assumptions and Notation

Hash values are modeled as iid random variables each with the distribution Uniform(0,1). S is the set of unique hash values maintained by the algorithm. k is the user-specified target value for $|S|$. It will be convenient to do the main analysis assuming that the hash values in the input stream are already unique. At the end, adjustments will be made to handle the possible presence of duplicates. Then $\mathcal{H}$ is the input stream of unique hash values h. The length of that stream is n. $\mathcal{H}$ is subdivided by the algorithm into a prefix of length w, which is used to initialize S, and and a suffix of length $u = n - w$, which is processed by the algorithm's main loop. During this processing, Line 10 is reached a certain number of times; this count will be refered to as the number of insertions, and will be denoted by the symbol t or the symbol i.

2.4 Analysis of Size of Sketch

This subsection contains an analysis of the size of S at line 4 of algorithm 1, that is, at the top of the main loop.

2.4.1 Expected Value

Because S was initialized with w hashes in line ?? and t hashes have been inserted subsequently in line ??, $|S| \leq w + t$. However, line ?? has also been executed t times, so $\theta = \alpha^t$, and S consists of "surviving hashes" $h < \alpha^t$. Let B be the random variable for the number of surviving hashes, and let $\hat{B}$ and B' be the corresponding random variables for surviving "initialization" hashes (that entered S via line ??) and "inserted hashes" (that entered S via line ??). Clearly $B = \hat{B} + B'$.

Now, since the initialization hashes are iid random draws from Uniform $[0, 1)$, the probability of each one surviving is α^t, and the probability distribution governing the number of surviving initialization hashes is Binomial(w, α^t).[2] Hence $E(\hat{B}|w, \alpha, t) = w \cdot \alpha^t$.

However, the number of surviving inserted hashes is not governed by a Binomial distribution, because each one has a different probability of surviving. Consider the most recent insert. Because it passed the test in line ??, it is a uniform deviate on the range

[2]Because the hashes are iid uniform deviates, their lifetimes in S are iid geometric deviates. Geometric distributions are memoryless.

$[0, \alpha^{t-1})$, so its survival probability is $\alpha^t/\alpha^{t-1} = \alpha^1$. Similarly the survival probability of the previously inserted hash is α^2, and so on back to the hash that was inserted first, whose survival probability is α^t. Hence $E(B'|\alpha, t) = \sum_{j=1}^{t} \alpha^j$, and

$$E(B|w,\alpha,t) = E(\hat{B}|w,\alpha,t) + E(B'|\alpha,t) \tag{1}$$

$$= w\cdot\alpha^t + \sum_{j=1}^{t}\alpha^j \tag{2}$$

$$= w\cdot\alpha^t + \frac{\alpha-\alpha^{t+1}}{1-\alpha} \tag{3}$$

After substituting in the recommended parameter values $w = k$ and $\alpha = k/(k+1)$, and doing a bit of algebra, one obtains:

$$\forall t,\ \ E(B|k, \frac{k}{k+1}, t) = k. \tag{4}$$

2.4.2 Variance

For generic w and α:

$$\sigma^2(B|w,\alpha,t) = \sigma^2(\hat{B}|w,\alpha,t) + \sigma^2(B'|\alpha,t) \tag{5}$$

$$= w\alpha^t(1-\alpha^t) + \sum_{j=1}^{t}\alpha^j(1-\alpha^j) \tag{6}$$

$$= w\ \alpha^t\ (1-\alpha^t) \tag{7}$$

$$+ \frac{\alpha + \alpha^{2(t+1)} - \alpha^{t+2} - \alpha^{t+1}}{1-\alpha^2} \tag{8}$$

For $w = k$ and $\alpha = k/(k+1)$:

$$\sigma^2(B) = \frac{\alpha-\alpha^{2t+1}}{1-\alpha^2} = \frac{\alpha}{1-\alpha^2}(1-\alpha^{2t}) \tag{9}$$

$$< \frac{\alpha}{1-\alpha^2} = \frac{k^2+k}{2k+1} \tag{10}$$

$$< \frac{k}{2} + \frac{1}{4}. \tag{11}$$

Corollary: for given n, k, and δ, there exists a constant c_b such that, with probability $1-\delta$, the size of S never exceeds $k \cdot c_b$ during the execution of the algorithm.

2.4.3 Distribution of Size of Sketch

Although the above expressions for $E(|S|)$ and $\sigma^2(|S|)$ are adequate for many purposes, the exact error analysis below requires the full probability distribution over possible values of $|S|$. Continuing with the above setup, for each $0 \le b \le (w+t)$ we need to know $\Pr(|S| = b \mid w, \alpha, t)$, that is, the probability that $|S| = b$ at the top of the loop in Algorithm ??, given that S was initialized with w hashes, and that t hashes have been subsequently inserted. Partition S into subsets $\hat{S}$ and S' which are respectively the surviving "initialization" hashes and the surviving "inserted" hashes, and let B, $\hat{B}$ and B' be the corresponding random variables. Then:

$$\Pr(B{=}b|w,\alpha,t) = \sum_{\hat{b}+b'=b}\Big(\Pr(\hat{B}{=}\hat{b}|w,\alpha,t) + \Pr(B'{=}b'|\alpha,t)\Big) \tag{12}$$

$$\Pr(\hat{B}{=}\hat{b}|w,\alpha,t) = \text{BinomialPDF}(\hat{b}; w, \alpha^t). \tag{13}$$

As mentioned above, the distribution $\Pr(B' = b' \mid \alpha, t)$ is not binomial because each of the t "inserted" hashes has a different survival probability. However, the distribution does obey the following recurrences, which can be efficiently evaluated using dynamic programming.

$$\Pr(B'{=}0\ |\alpha,0) = 1 \tag{14}$$

$$\Pr(B'{=}b'|\alpha,0) = 0 \tag{15}$$

$$\Pr(B'{=}0\ |\alpha,t) = (1-\alpha^t)\cdot\Pr(B'{=}0|\alpha,t{-}1) \tag{16}$$

$$\Pr(B'{=}b'|\alpha,t) = (1-\alpha^t)\cdot\Pr(B'{=}b'|\alpha,t{-}1) + \tag{17}$$

$$\alpha^t\cdot\Pr(B'{=}b'{-}1|\alpha,t{-}1) \tag{18}$$

2.5 Distribution of Estimates

Let X be the random variable for the estimate produced by the algorithm. This estimate can only assume a finite number of different values, each of the form $b/(\alpha^i)$, where b is a possible size of S, and i is a possible number of insertion operations. Let B and I be random variables governing those two quantities. Then:

$$\Pr(X = x|\alpha,w,u) = \sum_{(b,i)|x=b/\alpha^i}\Pr((B{=}b \wedge I{=}i)|\alpha,w,u).$$

and, by the chain rule for probabilities:

$$\Pr((B{=}b \wedge I{=}i)|\alpha,w,u) = \Pr(B{=}b|\alpha,w,u,i)\cdot\Pr(I{=}i|\alpha,w,u)$$

The distribution $\Pr(B{=}b|\alpha,w,u,i)$ was analysed in the previous section. The distribution $\Pr(I{=}i|\alpha,w,u)$ obeys the following recurrences, which can be efficiently evaluated using dynamic programming:

$$\Pr(I{=}0|\alpha,0) = 1 \tag{19}$$

$$\Pr(I{=}i|\alpha,0) = 0 \tag{20}$$

$$\Pr(I{=}0|\alpha,u) = (1-\alpha^i)\cdot\Pr(I{=}0|\alpha,u{-}1) \tag{21}$$

$$\Pr(I{=}i|\alpha,u) = (1-\alpha^i)\cdot\Pr(I{=}i|\alpha,u{-}1) + \tag{22}$$

$$\alpha^{i-1}\cdot\Pr(I{=}i{-}1|\alpha,u{-}1) \tag{23}$$

2.5.1 Technical Lemmas

This subsection contains two versions of a technical lemma that will be used in the subsequent analysis of the mean, variance, and higher moments of the distribution of estimates. First we will prove a simple version of the lemma which applies to the mean. Then we will prove a generalized version of the lemma which applies to the variance and higher moments of the distribution.

Simple Lemma: The function

$$f(k,u) = \sum_{i=0}^{u}\frac{1}{\alpha^i}\Pr(i|u)$$

satisfies the following base case and recurrence:

$$f(k,0) = 1 \tag{24}$$

$$f(k,u{+}1) = f(k,u) + \left(\frac{1-\alpha}{\alpha}\right)\cdot 1 \tag{25}$$

PROOF. The base case is true by inspection. The claimed recurrence is derived from equation ?? as follows:

$$f(k,u{+}1) = \sum_{i=0}^{u+1}\frac{1}{\alpha^i}\Pr(i|u+1) \tag{26}$$

$$= \left[\sum_{i=0}^{u}\frac{1}{\alpha^i}(1-\alpha^i)\Pr(i|u)\right] + 0 \tag{27}$$

$$+ 0 + \left[\sum_{i=1}^{u+1}\frac{1}{\alpha^i}(\alpha^{i-1})\Pr(i{-}1|u)\right] \tag{28}$$

Now we will evaluate the two bracketed sums, repeatedly noting that probability distributions sum to 1. For the first bracketed sum:

$$\sum_{i=0}^{u} \frac{1}{\alpha^i}\Pr(i|u) - \sum_{i=0}^{u} \frac{\alpha^i}{\alpha^i}\Pr(i|u) = f(k,u) - 1 \qquad (29)$$

Next we evaluate the second bracketed sum, after performing the change of variables $j = i-1$:

$$\sum_{j=0}^{u} \frac{\alpha^j}{\alpha^{j+1}}\Pr(j|u) = \frac{1}{\alpha}\cdot 1 \qquad (30)$$

Finally, combining ?? and ?? yields the claimed result:

$$f(k,u+1) = f(k,u) - 1 + \frac{1}{\alpha}\cdot 1 = f(k,u) + \left(\frac{1-\alpha}{\alpha}\right)\cdot 1 \quad (31)$$

□

Corollary of Simple Lemma: For all $u \in Z^+$:

$$f(k,u) = \frac{k+u}{k} \qquad (32)$$

PROOF. $\frac{k+u}{k}$ satisfies the same base case and recurrence as $f(k,u)$. The base case (eqn ??) can be verified by inspection. The recurrence (eqn ??) can be verified as follows:

$$\frac{k+(u+1)}{k} = \frac{k+u}{k} + \frac{1-\alpha}{\alpha}\cdot 1 = \frac{k+u}{k} + \frac{1}{k}. \qquad (33)$$

□

Generalized Lemma: If $g(q,k,u) = \sum_{i=0}^{u} \frac{1}{\alpha^{qi}}\Pr(i|u)$, then

$$\begin{aligned} g(0,k,u) &= 1 \qquad [\textit{by inspection}] && (34)\\ g(q,k,0) &= 1 \qquad [\textit{by inspection}] && (35)\\ g(q,k,u+1) &= g(q,k,u) + \left(\frac{1-\alpha^q}{\alpha^q}\right)\cdot g(q-1,k,u) && (36)\end{aligned}$$

PROOF. Omitted. This proof is basically the same as the proof of the Simplified Lemma. It is just necessary to verify that the exponents all work out as claimed. □

Corollaries of Generalized Lemma: For all $u \in Z^+$:

$$\begin{aligned} g(0,k,u) &= 1 && (37)\\ g(1,k,u) &= \frac{k+u}{k} \quad [= f(k,u)] && (38)\\ g(2,k,u) &= \frac{k^3 + 2k^2u + ku^2 + u(u-1)/2}{k^3} && (39)\end{aligned}$$

PROOF. $\frac{k^3+2k^2u+ku^2+u(u-1)/2}{k^3}$ satisfies the same base case and recurrence as $g(2,k,u)$. The base case (eqn ??) can be verified by inspection. The recurrence (eqn ??) can be verified as follows:

$$\begin{aligned} & g(2,k,u) + \left(\frac{1-\alpha^2}{\alpha^2}\right)\cdot g(1,k,u) && (40)\\ =& g(2,k,u) + \frac{1}{\alpha^2}\cdot\frac{k+u}{k} - \frac{k+u}{k} && (41)\\ =& g(2,k,u) + \frac{(k+1)^2}{k^2}\cdot\frac{k+u}{k} - \frac{k+u}{k} && (42)\\ =& g(2,k,u) + \frac{2k^2+k+2uk+u}{k^3} && (43)\\ =& \frac{k^3+2k^2u+2k^2+ku^2+2uk+k+\frac{1}{2}u^2+\frac{1}{2}u}{k^3} && (44)\\ =& \frac{k^3+2k^2(u+1)+k(u+1)^2+(u+1)u/2}{k^3} && (45)\\ =& g(2,k,u+1). && (46)\end{aligned}$$

□

2.5.2 Expected Value of Estimate

In this subsection it is proved that the new (algorithm+estimator) combination is unbiased. That is, if Algorithm 1 is run with parameters $w = k$ and $\alpha = k/(k+1)$ on a stream containing n unique values, and if the estimator $X = \frac{b}{\alpha^i}$ is used, then $E(X) = n$.
Theorem:

$$E(X|\alpha,w,u) = n. \qquad (47)$$

PROOF.

$$\begin{aligned} & E(X|\alpha,w,u) && (48)\\ =& \sum_{i=0}^{u}\sum_{b=0}^{w+i}\frac{b}{\alpha^i}\cdot\Pr(B{=}b|\alpha,w,i)\cdot\Pr(I{=}i|\alpha,u) && (49)\\ =& \sum_{i=0}^{u}\frac{1}{\alpha^i}\Pr(I{=}i|\alpha,u)\cdot\sum_{b=0}^{w+i} b\cdot\Pr(B{=}b|\alpha,w,i) && (50)\\ =& k\cdot\sum_{i=0}^{u}\frac{1}{\alpha^i}\Pr(I{=}i|\alpha,u) \quad \text{By Theorem ?} && (51)\\ =& k\cdot\frac{k+u}{k} \quad \text{By Lemma ?} && (52)\\ =& k+u = n. && (53)\end{aligned}$$

□

2.5.3 Variance of Estimate

[To improve formatting, conditioning on α will be implied, and $\Pr(Z = z|\ldots)$ will be abbreviated as $\Pr(z|\ldots)$].
Theorem:

$$\sigma^2(X) = \frac{k^2u + ku^2 + u(u-1)/2}{k^2}. \qquad (54)$$

PROOF. $\sigma^2(X) = E(X^2) - E^2(x)$. From eqn ?? we get $E^2(x) = n^2 = (k+u)^2$. The next lemma provides the value of $E(X^2)$. The result follows by algebra. □

Lemma:

$$E(X^2) = \frac{k^2u + ku^2 + u(u-1)/2 + k^4 + 2k^3u + k^2u^2}{k^2}. \qquad (55)$$

PROOF.

$$
\begin{aligned}
&E(X^2|w,u) && (56)\\
=&\sum_{i=0}^{u}\sum_{b=0}^{w+i}\left(\frac{b}{\alpha^i}\right)^2 \Pr(b|w,i)\ \Pr(i|u) && (57)\\
=&\sum_{i=0}^{u}\frac{1}{\alpha^{2i}}\Pr(i|u)\sum_{b=0}^{w+i} b^2\Pr(b|w,i) && (58)\\
=&\sum_{i=0}^{u}\frac{1}{\alpha^{2i}}\Pr(i|u)\left(\frac{\alpha-\alpha^{2i+1}}{1-\alpha^2}+k^2\right) && (59)\\
=&\frac{1}{1-\alpha^2}\left[\alpha\sum_{i=0}^{u}\frac{1}{\alpha^{2i}}\Pr(i|u)-\sum_{i=0}^{u}\alpha\Pr(i|u)\right] && (60)\\
&\quad + k^2\sum_{i=0}^{u}\frac{1}{\alpha^{2i}}\Pr(i|u) && (61)\\
=&\frac{1}{1-\alpha^2}\left[\alpha\cdot g(2,k,u)-\alpha\cdot 1\right]+k^2 g(2,k,u) && (62)\\
=&\left(\frac{\alpha}{1-\alpha^2}+k^2\right)\cdot g(2,k,u)-\frac{\alpha}{1-\alpha^2} && (63)\\
=&\frac{k(2k^2+2k+1)}{2k+1}\cdot g(2,k,u)-\frac{k(k+1)}{2k+1} && (64)\\
=&\frac{\begin{pmatrix} 2k^5+4k^4u+2k^3u^2+3k^2u^2+k^4+4k^3u \\ +2ku^2+k^2u-ku+u(u-1)/2 \end{pmatrix}}{k^2(2k+1)} && (65)\\
=&\frac{k^2u+ku^2+u(u-1)/2+k^4+2k^3u+k^2u^2}{k^2}. && (66)
\end{aligned}
$$

□

Corollary: Rewrite eqn ?? in terms of k and n, then let $n \to \infty$.

$$
\begin{aligned}
\sigma^2(X) =&\frac{(2k+1)n^2-(2k^2+2k+1)n+(k^2+k)}{2k^2} && (67)\\
\to&\frac{(2k+1)n^2}{2k^2}=\frac{n^2}{k-\frac{k}{2k+1}}<\frac{n^2}{k-\frac{1}{2}}. && (68)
\end{aligned}
$$

2.5.4 *Conjectured Third Moment*

Based on examples obtained via exact calculations of the full probability distribution of estimates for various values of k and u, we conjecture that the (centered) third moment of the distribution has denominator = k^5, and numerator =

$$
\begin{aligned}
&-\frac{1}{1}k^5u^1+\frac{3}{2}k^4u^2+\frac{5}{2}k^3u^3\\
&-\frac{3}{2}k^4u^1-\frac{3}{1}k^3u^2+\frac{5}{2}k^2u^3\\
&+\frac{3}{2}k^3u^1-\frac{5}{1}k^2u^2+\frac{5}{6}k^1u^3\\
&+\frac{5}{2}k^2u^1-\frac{2}{1}k^1u^2+\frac{1}{6}k^0u^3\\
&+\frac{7}{6}k^1u^1-\frac{1}{2}k^0u^2\\
&+\frac{1}{3}k^0u^1
\end{aligned}
$$

```
(-1.000000000 k^5 u^1) +
 (1.500000000 k^4 u^2) +
 (2.500000000 k^3 u^3) +

(-1.500000000 k^4 u^1) +
(-3.000000000 k^3 u^2) +
 (2.500000000 k^2 u^3) +

 (1.500000000 k^3 u^1) +
(-5.000000000 k^2 u^2) +
 (0.833333333 k^1 u^3) +      /*   5/6   */

 (2.500000000 k^2 u^1) +
(-2.000000000 k^1 u^2) +
 (0.166666667 k^0 u^3) +      /*   1/6   */

 (1.166666666 k^1 u^1) +      /*   7/6   */
(-0.500000000 k^0 u^2) +
 (0.333333335 k^0 u^1)
```

2.6 Three Variant Algorithms

Actually, there are now three different variants of our alpha algo + estimator scheme. All of them basically start out by running the alpha algo as described. However, a new variant estimates the cardinality of the stream to be $Z = \frac{k}{\alpha^i}$, where k is the configured target size for the sketch, in place of the actual number of items in the sketch. Another new variant can be modeled as making two passes. In the first pass, it runs the alpha algo and obtains a threshold $\theta = \alpha^i$. In the second pass, it re-processes the stream,[3] using a second, independent hash function, counting the number of hash values that are less than θ. If that count is c, then this scheme's estimate is $Y = \frac{c}{\alpha^i}$. The third variant is the version of the alpha algo that we actually run. It makes one pass over the stream, using a single hash function, producing a set S of samples whose size is b. The estimator is then $X = \frac{b}{\alpha^i}$. Now we collect together the expected values and variances of X, Y, and Z, and also KMV. There also some inequalities, which are true for big enough n/k.

$$
\begin{aligned}
E(Z) =&E(Y)=E(X)=E(KMV)=n && (69)\\
\sigma^2(Z) =&\frac{u(u-1)}{2k}=\frac{n^2-2nk+k^2-n+k}{2k} && (70)\\
<&\frac{n^2}{2k} && (71)\\
\sigma^2(Y) =&\frac{n^2-nk}{k} && (72)\\
<&\frac{n^2}{k} && (73)\\
\sigma^2(X) =&\frac{k^2u+ku^2+u(u-1)/2}{k^2} && (74)\\
=&\frac{(2k+1)n^2-(2k^2+2k+1)n+(k^2+k)}{2k^2} && (75)\\
<&\frac{n^2}{k-\frac{1}{2}} && (76)\\
\sigma^2(KMV) =& \frac{n^2-kn+n}{k-2}<\frac{n^2}{k-2} && (77)
\end{aligned}
$$

2.7 Analysis of the Estimator Z

[3]Actually, the two passes can be executed simultaneously. However, there *is* a factor of 2 cost in both time and space over the single-pass scheme which produces the estimator X.

$$E(Z|k,u) = \sum_{i=0}^{u} \frac{k}{\alpha^i} \cdot \Pr(i|u) \tag{78}$$

$$= k \cdot g(1,k,u) = k \cdot \frac{k+u}{k} = n \tag{79}$$

$$E(Z^2|k,u) = \sum_{i=0}^{u} \frac{k^2}{\alpha^{2i}} \cdot \Pr(i|u) \tag{80}$$

$$= k^2 \cdot g(2,k,u) \tag{81}$$

$$= \frac{k^3 + 2k^2u + ku^2 + u(u-1)/2}{k} \tag{82}$$

$$= \frac{u(u-1)/2}{k} + n^2 \tag{83}$$

$$\sigma^2(Z|k,u) = E(Z^2|k,u) - n^2 = \frac{u(u-1)}{2k}. \tag{84}$$

2.8 Analysis of the Estimator Y

$$E(Y|k,u) \tag{85}$$

$$= \sum_{i=0}^{u} \sum_{c=0}^{n} \frac{c}{\alpha^i} \cdot \Pr(c|n,i) \cdot \Pr(i|u) \tag{86}$$

$$= \sum_{i=0}^{u} \frac{1}{\alpha^i} \Pr(i|u) \cdot \sum_{c=0}^{n} c \text{ BinomialPMF}(c|n,\alpha^i) \tag{87}$$

$$= \sum_{i=0}^{u} \frac{1}{\alpha^i} \Pr(i|u) \cdot n\, \alpha^i \tag{88}$$

$$= n \cdot \sum_{i=0}^{u} \Pr(i|u) = n. \tag{89}$$

$$E(Y^2|k,u) \tag{90}$$

$$= \sum_{i=0}^{u} \sum_{c=0}^{n} \frac{c^2}{\alpha^{2i}} \cdot \Pr(c|n,i) \cdot \Pr(i|u) \tag{91}$$

$$= \sum_{i=0}^{u} \frac{1}{\alpha^{2i}} \Pr(i|u) \cdot \sum_{c=0}^{n} c^2 \text{ BinomialPMF}(c|n,\alpha^i) \tag{92}$$

$$= \sum_{i=0}^{u} \frac{1}{\alpha^{2i}} \Pr(i|u) \left[n\alpha^i - n\alpha^{2i} + n^2\alpha^{2i} \right] \tag{93}$$

$$= n \sum_{i=0}^{u} \frac{1}{\alpha^i} \Pr(i|u) - n \sum_{i=0}^{u} \Pr(i|u) + n^2 \sum_{i=0}^{u} \Pr(i|u) \tag{94}$$

$$= n \cdot g(1,k,u) - n + n^2 = \frac{n^2}{k} - n + n^2. \tag{95}$$

line ?? is because $E(c^2) = \sigma^2(c) + E^2(c) = np(1-p) + (np)^2$.

$$\sigma^2(Y|k,u) = E(Y^2|k,u) - n^2 = \frac{n^2}{k} - n. \tag{96}$$

Interestingly, this is the same variance as the fixed threshold algorithm run with $\theta = \frac{k}{n}$. However, the two distributions are *not* actually the same, despite having the same mean and variance.

2.8.1 Old Conjecture Section

For any specific combination of n and k, the above analysis allows us to calculate the full probability distribution over all estimates which could possibly be produced by Algorithm ??. Given the full probability distribution, one can compute summary statistics such as the mean and variance. When this calculation is done using infinite precision rational arithmetic, one finds that even though the intermediate probabilities require a huge number of digits to express, massive cancellations occur during the calculation of the mean and variance, resulting in an answer that is an integer or a fraction whose numerator and denominator contain just a few digits. We have found this to be true for thousands of (k,n) pairs. A few examples are contained in the following table:

k	u	n	$E(X)$	$\sigma^2(X)$
1	2	3	3	7 / 1
2	3	5	5	33 / 4
3	5	8	8	130 / 9
4	7	11	11	329 / 16
5	13	18	18	1248 / 25
6	23	29	29	4255 / 36
7	31	38	38	8711 / 49
8	43	51	51	18447 / 64
9	47	56	56	24769 / 81
10	59	69	69	42421 / 100

2.8.2 Ordering Issue

There is one respect in which this algorithm (and perhaps related algorithms like Gibbons) is theoretially weaker than KMV, Giroire, and HLL. Those latter algorithms are completely indifferent to the order in which the hash values are processed. They would produce the same results even if the input stream was collected, hashed, and sorted before being processed by the sketching algorithm.

On the other hand, the analysis of this paper's algorithm is based on the assumption that the hash values are being processed in random order. If the hash values were processed in a non-random order, then $E(|S|)$, $\sigma^2(|S|)$, and $\sigma^2(|X|)$ could all be different from the values calculated above. However, we conjecture that $E(X) = n$ would still hold.

Fortunately, this theoretical issue has no practical consequences in a true streaming context where each identifier is hashed and processed as soon as it arrives. That is because the hash function converts the arbitrarily ordered stream of identifiers into a randomly ordered stream of hash values.

2.9 Comparison with Ideal Fixed Threshold

Just for comparison purposes, we now analyze a procedure (IFT) that one couldn't actually run because it requires an oracle. The user specifies a target size k for the sketch. The oracle foresees what n will be, and supplies the fixed threshold $\theta = k/n$. Rather than fully exploiting the oracle's omniscience by returning the estimate k/θ, the algorithm processes the stream collecting into a set S all hash values less than θ, and finally outputs the estimate $|S|/\theta$.

The size of S produced by IFT is governed by a Binomial Distribution whose mean is k and whose variance is $\frac{nk-k^2}{n}$. For large n this approaches a Poisson Distribution whose mean is k and whose variance is k. That is bigger than the $\frac{k}{2} + \frac{1}{4}$ variance of this paper's method.

The estimates produced by IFT are governed by a "stretched" version of the same Binomial distribution, whose mean is n and whose variance is $\frac{n^2-nk}{k}$. For large n this approaches a "stretched" Poisson distribution whose mean is n and whose variance is $\frac{n^2}{k}$. That is smaller than the $\frac{n^2}{k-\frac{1}{2}}$ variance of this paper's method.

Also, the new scheme's full distribution of estimates is more skewed than that of the idealized scheme. See the plot.

2.10 Comparison with KMV

We will now compare with the KMV algorithm, which is analyzed in several papers, including Giroire. A KMV sketch always has size exactly k. Giroire proves that the expected value of the estimate is n, and that for large n the variance of the estimate is approximately $n^2/(k-2)$.

Compared to KMV, our new scheme has the same expected values for the sketch size and the estimate, and a larger variance for the sketch size, and a smaller variance for the estimate. Also, the new scheme's full distribution of estimates is less skewed than that of KMV. See the plot.

2.11 Cost of Executing the Algorithm

We will now analyze the execution cost of Algorithm 1, assuming that the set S is implemented using a hash table of size $c_h \cdot k$, and assuming that this hash table supports $O(1)$-cost lookup and insert operations.

Suppose that the stream contains D duplicate hashes in addition to the n unique hashes that we have been assuming so far. The cost of processing these is $O(D)$; this cost is incurred in lines 1-9.

Now, let us consider the $n = w + u$ unique hashes in the stream. These incur a cost of $O(n)$ in lines 1-9. In addition, t of the u unique hashes processed by the main loop make it past line 9 and cause the sketch to be updated in lines 10-12. We will not formally analyze the number of updates t, focusing instead on the cost of performing each update.[4]

Clearly, each update incurs an $O(1)$ cost in lines 10 and 11. However, line 12 is a problem. We want to delete from S those hashes that are not less than the newly reduced θ. Even if the hash table had an $O(1)$-cost deletion operation, we could not easily use it; without the priority queue we do not know of an inexpensive way to immediately locate the specific items that need to be deleted.

Our solution to this problem is to leave the "deleted" items in the hash table for a while. They are eventually removed during occasional rebuilds of the hash table. Each rebuild costs $O(k)$, so we need to make sure that the table isn't rebuilt more often than once per $O(k)$ inserts. Having ensured that, the lazy version of line 12 has an amortized cost of $O(1)$, and we are done.

2.11.1 *Further Discussion of the Lazy Algorithm*

Pseudocode for the lazy version of our algorithm is shown in figure 2. Instead of maintaining the set S exactly, this version maintains a set $T \supseteq S$ which not only contains S but also some other hashes that are bigger than θ but haven't been deleted yet by the next rebuild of the hash table which implements T.

Because the algorithm is tracking the larger set T but needs to return the size of S in line 6, some extra work is needed there; one must count the number of elements of T which are less than the current value of θ.

Now we will discuss the set size $c_r \cdot k$ which triggers a rebuild of the hash table implementing T. Recall that $c_b \cdot k$ is a high probability upper bound on the maximum size that S an attain during the execution of the algorithm, and that $c_h \cdot k$ is the number of slots in the actual hash table. The value of c_r needs to satisfy $c_b < c_r < c_h$, ideally with a large gap at each inequality.

The gap in the inequality $c_r < c_h$ ensures that the hash table's occupancy never becomes too large, thus improving the constant factor in the assumed $O(1)$ cost of hash table operations.

The gap in the inequality $c_b < c_r$ ensures that table rebuilds do not occur too often, thus improving the constant factor in the $O(1)$ amortized cost of line 12.

[4]Informally, $E(t)$ is roughly $k \ln(\frac{u+k}{k})$ for large k (assuming that $\alpha = \frac{k}{k+1}$). This is similar to the number of updates performed by the heap-based KMV algorithm.

Algorithm 2 Basic Algorithm with Lazy Deletion($\mathcal{H}, k, c_r$)

```
 1: $\alpha = k/(k+1)$.
 2: $T \leftarrow$ (First $k$ unique hashes in $\mathcal{H}$).
 3: $\theta \leftarrow 1.0$
 4: **Loop:**
 5: {Invariant: $T \supseteq S$}.
 6: **if** no more hashes in $\mathcal{H}$, **return** $(\theta, |\{(h < \theta) \in T\}|)$.
 7: $h \leftarrow$ next hash in $\mathcal{H}$.
 8: **if** $h \geq \theta$ **goto Loop**.
 9: **if** $h \in T$ **goto Loop**.
10: $T \leftarrow T \cup \{h\}$.
11: $\theta \leftarrow \alpha \cdot \theta$.
12: **if** $|T| >= c_r \cdot k$ **then** $T \leftarrow \{(h < \theta) \in T\}$.
13: **goto Loop.**
```

3. IMPLEMENTATION

Now that we have Algorithm 2, the main thing to decide is the specific kind of hash table to use in the actual implementation. The following factors influenced our decision.

1. Because the items to be stored are themselves hash values, a simple array of hash values accessed via open addressing is more space efficient than a library hash table that stores items, keys, and perhaps hashed keys.
2. Because we employ high quality hash functions (City Hash and Murmur Hash) that have good randomness in not only the high but also the low bits of the hash values, we can safely use open addressing with a power-of-two hash table size, thus allowing us to probe the table using masked-off low bits of the hash values.
3. Because our algorithm is already handling deletions lazily via occasional table rebuilds, the hash table does not need to support the efficient deletion of specific items. Hence we are free to use a double hashing version of open addressing which gracefully handles high load factors that permit the rebuilds to be done less often.

3.1 Optimizations

There is some redundant work in Algorithm 2 which can be avoided by using a single sequence of probes to achieve the combined effect of the lookup operation in line 9, and the possible insert in line 10.

As a further optimization, one can cause rebuilds to happen less often by modifying the implementation of the insert operation so that whenever possible it overwrites an "expired" item (that is, an h that is not less than the current threshold) instead of filling up a previously empty slot.

Pseudocode which reflects both of these optimizations appears as Algorithm 3. There are some tricky details involved in the implementation of the procedure OptimizedLookup().[5] In particular, because we are using double hashing, and because the overwriting insert operation treats expired hashes as if they were already deleted, the necessary sequence of probes for the lookup and for the insert are not quite the same. This is because the expired hashes are stepping stones which preserve the connectivity of the chains of probes leading to other hashes in the table.

Let h be the second argument to OptimizedLookup (). Then the lookup operation's sequence of probes cannot stop until it finds h or reaches a slot that is actually empty. However, the insert operation's

[5]Similar issues are thoroughly discussed in d in Section ?? of ??.

Algorithm 3 Optimized Algorithm With Lazy Deletion $(\mathcal{H}, k, c_r)$

```
1:  $\alpha = k/(k+1)$.
2:  $T \leftarrow$ (First $k$ unique hashes in $\mathcal{H}$).
3:  $\theta \leftarrow 1.0$
4:  **Loop:**
5:  {Invariant: $T \supseteq S$}.
6:  **if** no more hashes in $\mathcal{H}$, **return** $(\theta, |\{(h < \theta) \in T\}|)$.
7:  $h \leftarrow$ next hash in $\mathcal{H}$.
8:  **if** $h \geq \theta$ **goto Loop**.
9:  $(foundIt, InsertHere) \leftarrow$ OptimizedLookup$(T, h, \alpha \cdot \theta)$.
10: **if** $foundit$ **goto Loop**.
11: **if** Empty$(TableSlot[InsertHere])$ $\mathbf{then}$ $|T| \leftarrow |T|+1$.
12: $TableSlot[InsertHere] \leftarrow h$.
13: $\theta \leftarrow \alpha \cdot \theta$.
14: **if** $|T| >= c_r \cdot k$ **then** $T \leftarrow \{(h < \theta) \in T\}$.
15: **goto Loop.**
```

sequence of probes can stop upon reaching either an empty slot or a slot containing an expired hash.

Our method of obeying both of these rules with a single sequence of probes is to keep going until reaching h or an empty slot. However, during this sequence of probes, if the procedure notices any slot containing an expired hash, it remembers the first such slot. At the end of the sequence of probes, OptimizedLookup() returns (true,null) if it found h, or (false,locationOfFirstExpiredHash) if it noticed any expired hashes, else (false,locationOfEmptySlot).

One final detail is that we are calling OptimizedLookup() with the third argument $\alpha \cdot \theta$, so it considers any $h \geq \alpha \cdot \theta$ to be expired. This is anticipating the reduction in θ that will occur in line 13 on the code path which is executed in the event that h is not found in the table, and is therefore inserted, and therefore needs a slot to be inserted into. This code is correct even in the strange-seeming case where an h satisfying $\alpha \cdot \theta \leq h < \theta$ is inserted into the table, possibly overwriting another $\alpha \cdot \theta \leq h'$.