Draft of Theta-Sketch Paper

three authors

ABSTRACT

no abstract yet

1. STORY STRATEGY

In this version of the story, we don't need to make a big deal out of the simplicity claim. Instead we claim that our solution is just plain better than the obvious baselines in almost every way. What are those baselines?

- Beyer is the only true competitor. They described a KMV-based sketch mart with all of the needed capabilities. We beat it in almost every way. Less memory and CPU when creating base sketches. Lower variance for the base sketch estimator. Lower variance for the setops estimator.
- HLL is not in the running at all because the estimates for setops would be so bad.
- Not sure what to say about Chen Cao Bu.
- Although no one ever has spelled it out in these terms, it would be possible after reading Gibbons to come up with a sketch-mart scheme that is similar to ours, but with α = 1/2, and with a different rule for when to multiply θ by α. That hypothetical scheme is better than Beyer, but is not as good as our actual scheme.

2. BIG-DATA SKETCH MART: MOTIVATION AND REQUIREMENTS

Lee is working on this section.

3. OUR SOLUTION

Our sketch-mart solution has several main ingredients.

1. A Sketch Data Structure (θ, S).
2. A practical Algorithm X, whose inputs are a size target k and a stream $\mathcal{A}$ of size n, and whose outputs are a sketch (θ, S) and an estimate Z of n. These outputs have the following desirable properties: $E(|S|) = k$, $\sigma^2(|S|) < k/2 + 1/4$, $E(Z) = n$, and $\sigma^2(Z) < n^2/2k$. Furthermore, the amortized update time of Algorithm X is $O(1)$ with excellent constant factors.
3. A modified Algorithm Y, whose inputs are a size target k and a stream $\mathcal{A}$ of size n, and whose output is a sketch (θ, S') which has the theoretically convenient property that the value of θ is independent of the hash function which produced the elements of S'.
4. A rule for evaluating an arbitrary set expression over a collection of sketches, whose output is not a fact, but rather a sketch. This allows a sub-expression of a larger expression to be evaluated as soon as the data becomes available.
5. An estimator which is unbiased (that is, whose expected value is n) for a very wide class of sketches (θ, S). However, the estimator's variance depends on the algorithm that was used to create the sketch:
   (a) If the sketch was produced by KMV with size target k, then the estimator's variance is $\sigma^2 < n^2/(k-2)$.
   (b) If it was produced by Algorithm X with size target k, then the estimator's variance is $\sigma^2 < n^2/(k-(1/2))$.
   (c) If it was produced by Algorithm Y with size target k, then the estimator's variance is $\sigma^2 < n^2/k$
   (d) If the sketch (θ, S) was produced by the Set Expression Rule applied to a collection of sketches produced by Algorithm Y, then, conditioning on the sketch's particular value of θ, the estimator's variance is $\sigma^2 = (n/\theta) - n$.

4. NEW ALGORITHM

Because of the larger systems context, we will focus on algorithms which naturally[1] provide a co-ordinated sample of approximately k hash values that can be used to estimate the cardinality of a set expression evaluated over a collection of streams. We will call this family of algorithms the "KMV family", even though it includes some algorithms (i.e. Gibbons-style sampling algorithms) that are traditionally not lumped together with KMV. We note that this "KMV family" does not include HLL and its variants.

One of our main results will be a new sketching algorithm whose estimate of n has variance < $n^2/(2k)$. This is the lowest ever for

[1]Although they obviously can be constructed, we will not be discussing hybrids which e.g. subdivide the memory budget between an instance of KMV and an instance of HLL.

Algorithm 1 Basic Algorithm for Processing a Stream

```
 1: Inputs: (target size k, HashFunction(), StreamOfIdentifiers).
 2: StreamOfHashValues ← HashFunction(StreamOfIdentifiers).
 3: prefix ← shortest prefix of StreamOfHashValues containing
                k unique hash values.
 4: restOfStreamOfHashValues ← the corresponding suffix.
 5: S ← the k unique hash values in prefix.
 6: α ← k/(k + 1).
 7: i ← 0.
 8: for all h in restOfStreamOfHashValues do
 9:    {Invariant: E(|S|) = k}.
10:    if h < α^i AND h ∉ S then
11:       S ← S ∪ {h}.
12:       i ← i + 1.
13:       S ← S \ {(x ≥ α^i) ∈ S}.
14:    end if
15: end for
16: Output the estimate Z = k/α^i and the sketch (θ = α^i, S).
```

an algorithm belonging to the "KMV family". The algorithm is based on a novel counting process that is different those used in the actual KMV algorithm, or in KMV variants such as Giroire, or in Gibbons-style sampling algorithms, or in HLL for that matter.

We will call this the "Basic Algorithm". PseudoCode is shown in Figure ??. Its arguments are a size target "k", a hash function, and a stream of identifiers. Purely to reduce notational clutter in the pseudocode and its analysis, we pretend that the first step is to apply the hash function to the stream of identifiers, producing a stream of hashes. The next three lines are a high level description of the algorithm's initialization phase, which processes just enough of the stream to find the first k unique hash values, which are used to initialize the "sample" set S. The remainder of the stream of hashes is then processed by the remainder of the program.

4.1 Overview Of Analysis

We will be proving a number of theorems about the basic algorithm, and also about a couple of variants (not mentioned yet) of the basic algorithm.

In section ??, we will lay some groundwork in the form of a technical lemma about a function $g()$ which encodes the behavior of several summations involving the probability distribution which governs the value of i during the execution of the basic algoritm.

In section ?? we will use this groundwork to analyze the estimate $Z = k/\alpha^i$. We will prove that it is unbiased, that its variance is ??, the best ever for an algorithm from the "KMV familty". In fact, we will prove that all of the moments of its distribution are well behaved, resulting in a strong bound of the epsilon-delta variety.

In section ?? we will prove that the invariant mentioned in line ?? of the "Basic Algorithm" is indeed true. That is, $E(|S|) = k$ at the top of the "for loop". We will also bound the variance of the size of S. These results are needed to show the memory and CPU resources consumed by this algorithm are roughly similar to (or better than) other algorithms in the KMV family.

In section ?? we will analyze a different estimator $X = |S|/\alpha^i$ which is related to the "practical" version of our setops scheme. This analysis will re-use the technical lemma about the function "$g()$", and will also re-use the analysis of the size of the set S.

In section ?? we will analyze yet another estimator Y which is related to the "theoretical" version of our setops scheme. This analysis will re-use the technical lemma about the function "$g()$".

4.2 Groundwork for the Analysis

In this section we lay the groundwork for the next section's analysis of the estimator $Z = k/\alpha^i$.

We need to establish some assumptions and some notation. For the purposes of the analysis, we are assuming that the hash function produces perfectly random real values on the interval $[0, 1)$. Hence the probability of a hash collision is zero. The symbol n denotes the number of unique identifiers in the input stream. All of the algorithms in this section handle the first k unique identifiers during an initialization phase. Hence the number of unique identifers processed by the main loop of each algorithm is $u = n - k$. The symbol α always denotes the fraction $k/(k + 1)$.

4.2.1 The distribution Pr(i | u)

Now, let us analyze the probability distribution $Pr(i|u)$ which governs the distribution of the final value of i when one runs the basic algorithm on a stream containing $n = k + u$ unique identifiers. We will characterize this distribution via a recurrence and base cases. The base cases are easy to see. If $u = 0$, then the algorithm's "for loop" is skipped, and the final value of i is zero. Hence $Pr(i = 0|u = 0) = 1$, and $Pr(i|u = 0) = 0, \forall i > 0$. On the other hand, if $u > 0$, then the body of the "for loop" will be executed at least once, and the test in line ?? will succeeed at least once (because a random value from $[0, 1)$ must be less than $1 = \alpha^0$). Therefore $i \geq 1$, so for all $u > 0$, $Pr(i = 0|u) = 0$.

The recurrence can be worked out by considering a state space in which the pair (i, u) means that i increments have occured, and u unique hash values have been processed.

Suppose that the algorithm ends up in state (i, u). Before processing the final unique hash value and reaching this state, the algorithm must have been in one of two states: either $(i, u - 1)$, or $(i - 1, u - 1)$. We will make the induction hypothesis that we already know the probabilities of reaching those two states.

Now we need to know the probability of transitioning from $(i, u-1)$ to (i, u), and the probability of transitioning from $(i-1, u-1)$ to (i, u).

A transition from $(i, u - 1)$ to (i, u) occurs when the test in line ?? fails, that is, when $h \geq \alpha^i$. Because h is a uniform random variable on $[0, 1)$, this happens with probability $(1-\alpha^i)$. Hence the probability of reaching state (i, u) by going through state $(i, u-1)$ is $(1 - \alpha^i)$ times the probability of reaching state $(i, u - 1)$.

A transition from $(i - 1, u - 1)$ to (i, u) occurs when the test in line ?? succeeds, that is, when $h < \alpha^{i-1}$. Because h is a uniform random variable on $[0, 1)$, this happens with probability α^{i-1}. Hence the probability of reaching state (i, u) by going through state $(i - 1, u - 1)$ is $(1 - \alpha^i)$ times the probability of reaching state $(i - 1, u - 1)$.

The total probability of reaching (and hence ending up in) state (i, u) is then the sum of the probabilities of reaching it via these two routes. This establishes the recurrence, which we now explicitly state along with the above-mentioned base cases:

$$\Pr(0|0) = 1 \tag{1}$$

$$\Pr(i|0) = 0, \quad \forall i > 0 \tag{2}$$

$$\Pr(0|u) = 0, \quad \forall u > 0 \tag{3}$$

$$\Pr(i|u) = (1 - \alpha^i) \cdot \Pr(i|u-1) + \tag{4}$$

$$\alpha^{i-1} \cdot \Pr(i-1|u-1), \quad \forall i > 0, \forall u > 0 \tag{5}$$

4.2.2 The Function g(q,k,u)

The analysis of the Estimator Z involves several summations involving the probability distribution $Pr(i|u)$. As part of the groundwork for that analysis, we will prove a technical lemma about a function $g(q, k, u)$ which encodes the values of those summations. This section is completely technical and can be safely skipped by readers who are more interested in the high-level story than in the

details of the proofs. Here is the definition of $g(q,k,u)$:

$$g(q,k,u) = \sum_{i=0}^{u} \frac{1}{\alpha^{qi}} \Pr(i|u) \tag{6}$$

Technical Lemma: The function $g(q,k,u)$ defined by equation ?? satisfies the following base cases and recurrence:

$$g(0,k,u) = 1 \tag{7}$$
$$g(q,k,0) = 1 \tag{8}$$
$$g(q,k,u+1) = g(q,k,u) + \left(\frac{1-\alpha^q}{\alpha^q}\right) \cdot g(q-1,k,u) \tag{9}$$

PROOF. The base cases can be verified by inspection. The recurrence can be derived from the recurrence for $Pr(i|u)$ as follows:

$$g(q,k,u+1) = \sum_{i=0}^{u+1} \frac{1}{\alpha^{qi}} \Pr(i|u+1) \tag{10}$$
$$= \left[\sum_{i=0}^{u} \frac{1}{\alpha^{qi}} (1-\alpha^i) \Pr(i|u)\right] + 0 \tag{11}$$
$$+ 0 + \left[\sum_{i=1}^{u+1} \frac{1}{\alpha^{qi}} (\alpha^{i-1}) \Pr(i-1|u)\right] \tag{12}$$

Now we will consider the two bracketed sums. For the first one:

$$\sum_{i=0}^{u} \frac{1}{\alpha^{qi}} (1-\alpha^i) \Pr(i|u) \tag{13}$$
$$= \sum_{i=0}^{u} \frac{1}{\alpha^{qi}} \Pr(i|u) - \sum_{i=0}^{u} \frac{\alpha^i}{\alpha^{qi}} \Pr(i|u) \tag{14}$$
$$= g(q,k,u) - \sum_{i=0}^{u} \frac{1}{\alpha^{(q-1)i}} \Pr(i|u) \tag{15}$$
$$= g(q,k,u) - g(q-1,k,u). \tag{16}$$

For the second one:

$$\sum_{i=1}^{u+1} \frac{1}{\alpha^{qi}} (\alpha^{i-1}) \Pr(i-1|u) \tag{17}$$
$$= \sum_{j=0}^{u} \frac{1}{\alpha^{q(j+1)}} (\alpha^j) \Pr(j|u) \tag{18}$$
$$= \frac{1}{\alpha^q} \sum_{j=0}^{u} \frac{1}{\alpha^{(q-1)j}} \Pr(j|u) \tag{19}$$
$$= \frac{1}{\alpha^q} g(q-1,k,u). \tag{20}$$

Adding ?? and ?? yields the claimed result:

$$g(q,k,u) - g(q-1,k,u) + \frac{1}{\alpha^q} g(q-1,k,u) \tag{21}$$
$$= g(q,k,u) + \left(\frac{1-\alpha^q}{\alpha^q}\right) \cdot g(q-1,k,u) \tag{22}$$

□

Corollaries of Technical Lemma: For all integers $u \geq 0$:

$$g(1,k,u) = \frac{k+u}{k} \tag{23}$$
$$g(2,k,u) = \frac{k^3 + 2k^2u + ku^2 + u(u-1)/2}{k^3} \tag{24}$$

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

$$\frac{k+(u+1)}{k} = \frac{k+u}{k} + \frac{1-\alpha}{\alpha} \cdot 1 = \frac{k+u}{k} + \frac{1}{k}. \tag{25}$$

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

$$g(2,k,u) + \left(\frac{1-\alpha^2}{\alpha^2}\right) \cdot g(1,k,u) \tag{26}$$
$$= g(2,k,u) + \frac{1}{\alpha^2} \cdot \frac{k+u}{k} - \frac{k+u}{k} \tag{27}$$
$$= g(2,k,u) + \frac{(k+1)^2}{k^2} \cdot \frac{k+u}{k} - \frac{k+u}{k} \tag{28}$$
$$= g(2,k,u) + \frac{2k^2+k+2uk+u}{k^3} \tag{29}$$
$$= \frac{k^3 + 2k^2u + 2k^2 + ku^2 + 2uk + k + \frac{1}{2}u^2 + \frac{1}{2}u}{k^3} \tag{30}$$
$$= \frac{k^3 + 2k^2(u+1) + k(u+1)^2 + (u+1)u/2}{k^3} \tag{31}$$
$$= g(2,k,u+1). \tag{32}$$

□

4.3 Analysis of the Estimator Z

4.3.1 Mean and Variance

The expected value and variance of the estimator $Z = k/\alpha^i$ can be computed with the help of the above technical lemma:

$$E(Z|k,u) = \sum_{i=0}^{u} \frac{k}{\alpha^i} \cdot \Pr(i|u) \tag{33}$$
$$= k \cdot g(1,k,u) = k \cdot \frac{k+u}{k} = n \tag{34}$$
$$E(Z^2|k,u) = \sum_{i=0}^{u} \frac{k^2}{\alpha^{2i}} \cdot \Pr(i|u) \tag{35}$$
$$= k^2 \cdot g(2,k,u) \tag{36}$$
$$= \frac{k^3 + 2k^2u + ku^2 + u(u-1)/2}{k} \tag{37}$$
$$= \frac{u(u-1)/2}{k} + n^2 \tag{38}$$
$$\sigma^2(Z|k,u) = E(Z^2|k,u) - n^2 \tag{39}$$
$$= \frac{u(u-1)}{2k} \tag{40}$$
$$= \frac{n^2 - 2nk + k^2 - n + k}{2k} \tag{41}$$
$$< \frac{n^2}{2k} \quad \text{for large } \frac{n}{k}. \tag{42}$$

4.3.2 Discussion

The variance of Estimator Z is smaller by a factor of 2 than that of other "KMV class" algorithms such as KMV itself, Giroire's generalizations of KMV, various algorithms that sample using a decreasing threshold such as Gibbons and our our own estimator X, and even sampling with the "ideal" fixed fixed threshold $\theta = k/n$. But why, exactly? We do not know yet.

4.3.3 Higher Moments and Confidence Intervals

Anirban will produce this material.

4.4 Analysis of the Size of S

We will now show that the expected value of $|S|$ is k, and its variance is upper bounded by $k/2 + 1/4$. These results are needed to establish that our basic algorithm is practical, and also to show that estimator Z is achieving lower variance than other KMV-like algorithms while collecting roughly the same number of hash values in its sample.

4.4.1 Intuitive Discussion

Before getting to the actual analysis, we mention that the variance of $|S|$ is smaller than one might expect from a couple of plausible-sounding but inaccurate analogies for what is going on.

The first inaccurate analogy is that the mechanism governing the size of S resembles sampling with a fixed threshold $\theta = k/n$. However, for $n \gg k$, that analogy suggests that $|S|$ has a Poisson distribution; the resulting variance of k is too big by a factor of 2.

The second inaccurate analogy assumes that the mechanism is essentially an ordinary random walk in which each step is caused by the combined effect of lines ?? and ??. However, that would cause the variance of $|S|$ to keep growing forever as n increases; that answer is wrong by an unbounded factor.

A third analogy is only a little bit inaccurate, namely that the size of S is governed by a certain AR(1) process, namely a random walk whose steps are determined by a linear restoring force plus noise. The linear restoring force arises from the fact that line ?? always inserts exactly one item, but the expected number of items deleted by line ?? is less than 1 if $|S| < k$, and more than 1 if $|S| > k$. When an approximate description of our process consisting of the actual restoring force plus an approximation of the noise distribution is plugged into a basic theorem about $AR(1)$ processes, an answer close to $k/2$ pops out for the "steady state" variance of the size of S. This is close to the true answer.

4.4.2 The Actual Analysis

In this section we will put aside the above intuitive arguments and do a proper analysis of the distribution governing $|S|$. This can be done by analyzing two survival processes, a process **K** governing the k items which were inserted into S by line ?? of Algorithm ??, and a process **I** governing the i items which were inserted into S by line ?? of the algorithm.

Process K: A hash value h inserted into S by line ?? ends up in the final set S if and only it survives the i rounds of deletion which subsequently occur in line ??. Since the test in line ?? becomes monotonically more stringent in successive deletion rounds, this will happen if and only h can survive the final round of deletion, where the test is $h < \alpha^i$. Because h is a uniform random variable on $[0, 1)$, this occurs with probability α^i. Furthermore, because the h's are iid uniform random variables, the indicator variables for their respective survivals are iid Bernoulli random variables, and the total number of survivors amongst the k initial members of S is governed by the Binomial Distribution with k draws and probability α^i. Hence the expected number of survivors of process **K** is $k \cdot \alpha^i$, and the variance is $k \cdot \alpha^i \cdot (1 - \alpha^i)$.

Process I: This process is not governed by a Binomial Distribution, because the survival probabilities of the i hashes inserted by line ?? are all different. In all cases, ultimate survival is determined by same final test $h < \alpha^i$ (in line ??); the explanation for the differing survival probabilities is that each hash value passed a different test (in line ??) upon entering the set S. For example, consider the final hash value to enter the set S. Because it passed the entry test $h < \alpha^{i-1}$ in line ??, it passes the final deletion test $h < \alpha^i$ with probability $\alpha^i/\alpha^{i-1} = \alpha^1$. Similarly, the second-to-last hash value to enter the set S passes the final test with probability $\alpha^i/\alpha^{i-2} = \alpha^2$. Overall, the survival of the i hash values can be modelled by i iid Bernoulli random variables whose respective probabilities are $\alpha^1, \alpha^2, \ldots, \alpha^i$. The expected number of survivors of process **I** is $\sum_{j=1}^{i} \alpha^j$, and the variance is $\sum_{j=1}^{i} \alpha^j(1 - \alpha^j)$.

Now, because all of the survival events are independent, the probability distribution governing the size of S is the sum of the probability distributions governing process **K** and process **I**. Therefore:

$$E(|S|) = k \cdot \alpha^i + \sum_{j=1}^{i} \alpha^j = k \cdot \alpha^i + \frac{\alpha - \alpha^{i+1}}{1 - \alpha} \tag{43}$$

$$= k. \tag{44}$$

$$\sigma^2(|S|) = k \cdot \alpha^i \cdot (1 - \alpha^i) + \sum_{j=1}^{i} \alpha^j(1 - \alpha^j) \tag{45}$$

$$= k\ \alpha^i\ (1 - \alpha^i) \tag{46}$$

$$+ \frac{\alpha + \alpha^{2(i+1)} - \alpha^{i+2} - \alpha^{i+1}}{1 - \alpha^2} \tag{47}$$

$$= \frac{\alpha - \alpha^{2i+1}}{1 - \alpha^2} \tag{48}$$

$$= \frac{\alpha}{1 - \alpha^2}(1 - \alpha^{2i}) \tag{49}$$

$$< \frac{\alpha}{1 - \alpha^2} \tag{50}$$

$$= \frac{k^2 + k}{2k + 1} \tag{51}$$

$$< \frac{k}{2} + \frac{1}{4}. \tag{52}$$

4.5 Analysis of the Estimator X

Our basic algorithm outputs not only the stand-alone estimator $Z = k/\alpha^i$, but also the theta sketch $(\theta = \alpha^i, S)$ which is stored in the sketch mart for later use in set expressions. Very roughly speaking, the accuracy of a set expression estimate is determined by combination of the estimator $X = |S|/\alpha^i$ and a penalty factor which grows worse as the true answer grows smaller relative to the size of the streams. We will discuss the penalty factor later; right now we will analyze the estimator X. We will prove that X is unbiased, and that the variance of X is slightly less than the variance of KMV.

In the following, B is the random variable for the size of $|S|$.

$$E(X|k, u) \tag{53}$$

$$= \sum_{i=0}^{u} \sum_{b=0}^{k+i} \frac{b}{\alpha^i} \cdot \Pr(B = b|k, i) \cdot \Pr(I = i|u) \tag{54}$$

$$= \sum_{i=0}^{u} \frac{1}{\alpha^i} \Pr(I = i|u) \cdot \sum_{b=0}^{k+i} b \cdot \Pr(B = b|k, i) \tag{55}$$

$$= \sum_{i=0}^{u} \frac{1}{\alpha^i} \Pr(I = i|u) \cdot k \tag{56}$$

$$= k \cdot g(1, k, u) = k \cdot \frac{k + u}{k} = k + u \tag{57}$$

$$= n. \tag{58}$$

$$E(X^2|k,u) \tag{59}$$

$$=\sum_{i=0}^{u}\sum_{b=0}^{k+i}\left(\frac{b}{\alpha^i}\right)^2 \Pr(b|k,i)\ \Pr(i|u) \tag{60}$$

$$=\sum_{i=0}^{u}\frac{1}{\alpha^{2i}}\Pr(i|u)\sum_{b=0}^{k+i}b^2\Pr(b|k,i) \tag{61}$$

$$=\sum_{i=0}^{u}\frac{1}{\alpha^{2i}}\Pr(i|u)\left(\frac{\alpha-\alpha^{2i+1}}{1-\alpha^2}+k^2\right) \tag{62}$$

$$=\frac{1}{1-\alpha^2}\left[\alpha\sum_{i=0}^{u}\frac{1}{\alpha^{2i}}\Pr(i|u)-\sum_{i=0}^{u}\alpha\Pr(i|u)\right] \tag{63}$$

$$+k^2\sum_{i=0}^{u}\frac{1}{\alpha^{2i}}\Pr(i|u) \tag{64}$$

$$=\frac{1}{1-\alpha^2}\left[\alpha\cdot g(2,k,u)-\alpha\cdot 1\right]+k^2 g(2,k,u) \tag{65}$$

$$=(\frac{\alpha}{1-\alpha^2}+k^2)\cdot g(2,k,u)-\frac{\alpha}{1-\alpha^2} \tag{66}$$

$$=\frac{k(2k^2+2k+1)}{2k+1}\cdot g(2,k,u)-\frac{k(k+1)}{2k+1} \tag{67}$$

$$=\frac{\left(\begin{array}{c}2k^5+4k^4u+2k^3u^2+3k^2u^2+k^4+4k^3u\\+2ku^2+k^2u-ku+u(u-1)/2\end{array}\right)}{k^2(2k+1)} \tag{68}$$

$$=\frac{k^2u+ku^2+u(u-1)/2+k^4+2k^3u+k^2u^2}{k^2}. \tag{69}$$

$$\sigma^2(X|k,u)=E(X^2|k,u)-E^2(X|k,u) \tag{70}$$

$$=\frac{k^2u+ku^2+u(u-1)/2}{k^2} \tag{71}$$

$$=\frac{(2k+1)n^2-(2k^2+2k+1)n+(k^2+k)}{2k^2} \tag{72}$$

$$\to\frac{(2k+1)n^2}{2k^2} \tag{73}$$

$$=\frac{n^2}{k-\frac{k}{2k+1}} \tag{74}$$

$$<\frac{n^2}{k-\frac{1}{2}}. \tag{75}$$

4.6 Analysis of the Estimator Y

Although our actual sketch mart system computes estimates for set expressions from theta sketches produced by the Basic Algorithm, and this works fine in practice (see the experiments section below), it appears that a formal analysis of the resulting errors would be very complicated.

So, strictly for the purposes of enabling a simple error analysis for set expressions, we now present a modified version of the basic algorithm, which produces a different theta sketch $(\theta_1=\alpha^i, S_2)$, and a corresponding estimator $Y=|S_2|/\theta_1$. We will prove that Y is unbiased, and that the variance of Y is slightly less than the variance of KMV.

The modification consists of a second pass over the stream, using a second hash function that is independent of the first hash function. During this second pass, we collect into the set S_2 all (second) hash values that are less than the threshold $\theta_1=\alpha^i$ that was computed using the first hash function.[2] This extra work ensures that the theta-sketch's threshold θ_1 and sample set S_2 are independent. This causes the size of S_2 to be governed by the Binomial distribution with n throws and probability θ_1, which greatly simplifies the error analysis for set expressions. First, though, we will analyse the single-sketch estimator $Y=|S_2|/\theta_1$.

In the following, C is the random variable for the size of $|S_2|$.

$$E(Y|k,u) \tag{76}$$

$$=\sum_{i=0}^{u}\sum_{c=0}^{n}\frac{c}{\alpha^i}\cdot\Pr(C=c|n,i)\cdot\Pr(i|u) \tag{77}$$

$$=\sum_{i=0}^{u}\frac{1}{\alpha^i}\Pr(i|u)\cdot\sum_{c=0}^{n}c\ \mathrm{BinomialPMF}(c|n,\alpha^i) \tag{78}$$

$$=\sum_{i=0}^{u}\frac{1}{\alpha^i}\Pr(i|u)\cdot n\ \alpha^i \tag{79}$$

$$=n\cdot\sum_{i=0}^{u}\Pr(i|u)=n. \tag{80}$$

$$E(Y^2|k,u) \tag{81}$$

$$=\sum_{i=0}^{u}\sum_{c=0}^{n}\frac{c^2}{\alpha^{2i}}\cdot\Pr(C=c|n,i)\cdot\Pr(i|u) \tag{82}$$

$$=\sum_{i=0}^{u}\frac{1}{\alpha^{2i}}\Pr(i|u)\cdot\sum_{c=0}^{n}c^2\ \mathrm{BinomialPMF}(c|n,\alpha^i) \tag{83}$$

$$=\sum_{i=0}^{u}\frac{1}{\alpha^{2i}}\Pr(i|u)\left[n\alpha^i-n\alpha^{2i}+n^2\alpha^{2i}\right] \tag{84}$$

$$=n\sum_{i=0}^{u}\frac{1}{\alpha^i}\Pr(i|u)-n\sum_{i=0}^{u}\Pr(i|u)+n^2\sum_{i=0}^{u}\Pr(i|u) \tag{85}$$

$$=n\cdot g(1,k,u)-n+n^2 \tag{86}$$

$$=\frac{n^2}{k}-n+n^2. \tag{87}$$

$$\sigma^2(Y|k,u)=E(Y^2|k,u)-n^2=\frac{n^2}{k}-n. \tag{88}$$

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.

5. IMPLEMENTATION OF BASIC ALGO

The writeup in this section is somewhat out of date.

5.1 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.

[2]It is not hard to figure out how to obtain the same output using a *single* pass using both hash functions. However, the resulting algorithm would still be twice as expensive as the unmodified basic algorithm in terms of both memory and CPU; that is why we don't actually do it in practice.

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.**

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.[3]

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.

5.1.1 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.

6. IMPLEMENTATION

[3]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.

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.

6.1 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().[4] 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 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 $\alpha \cdot \theta \leq h'$.

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

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

$\alpha = k/(k+1)$.
$T \leftarrow$ (First k unique hashes in $\mathcal{H}$).
$\theta \leftarrow 1.0$
Loop:
{Invariant: $T \supseteq S$}.
if no more hashes in $\mathcal{H}$, **return** $(\theta, |\{(h < \theta) \in T\}|)$.
$h \leftarrow$ next hash in $\mathcal{H}$.
if $h \geq \theta$ **goto Loop**.
$(foundIt, InsertHere) \leftarrow \text{OptimizedLookup}(T, h, \alpha \cdot \theta)$.
if $foundit$ **goto Loop**.
if $\text{Empty}(TableSlot[InsertHere])$ **then** $|T| \leftarrow |T|+1$.
$TableSlot[InsertHere] \leftarrow h$.
$\theta \leftarrow \alpha \cdot \theta$.
if $|T| >= c_r \cdot k$ **then** $T \leftarrow \{(h < \theta) \in T\}$.
goto Loop.

7. ANALYSIS OF Y ESTIMATOR

sig-alternate times algorithm, algorithmic amsmath, amssymb graphicx algorithm, algorithmic color

ABSTRACT

no abstract yet

8. NEW ANALYSIS

Suppose we have two streams A and B, and we define the joined stream as $A \Delta B$ where Δ is a set operator. We are trying to analyze the estimator Y. Let θ_a and θ_b be the threshold values obtained for each of the two streams using only the first pass and using two independent private hash functions h_a and h_b. θ_a and θ_b are random variables that are independent of each other. We then set $\theta_{ab} = \min(\theta_a, \theta_b)$.

For each element i of $A \Delta B$, let M_i be the indicator variable that indicates whether or not $h(i) < \theta_{ab}$, i.e. whether or not i is present in the composed sketch created from A and B. So

$$M_i = \begin{cases} 1 & \text{w.p. } \theta_{ab} \\ 0 & \text{else.} \end{cases}$$

Let $M = \sum_{i \in A \Delta B} M_i$. We then bound the moment generating function $E[\exp(tM)] = \prod_{i \in A \Delta B} E[\exp(tM_i)]$. Also,

$$E[\exp(tM_i)] = \sum_j E[\exp(tM_i) | \theta_{ab} = \alpha_j] P(j | u_a, u_b)$$

where $P(j|u_a, u_b)$ denotes the probability that $\theta_{ab} = \alpha^j$. Also,

$$E[\exp(tM_i) \mid \alpha^j] = 1 + \alpha^j (e^t - 1).$$

The following Lemma follows directly from adapting the Proposition 2 in [] to the case of arbitrary bases. For all i and u,

$$p_{iu} < \exp(-k) i (1 - \alpha^i)^u.$$

Hence, for $i < \ln(n) - \ln\ln n$,

Similarly, the following Lemma follows from Proposition 4 in []. For $i = 2\log(n) + \delta$, with $\delta \geq 0$, we have $p_{iu} = O(2^{-\delta} n^{-0.99})$.

Finally, we show the claim. Let bad denote the event that the estimate is $\epsilon |A \delta B|$ away from the the truth. WLOG, let $u_a \geq u_b$. $J_1 = [1, \log(u_a) - \log\log(u_a)]$,

$$\Pr[\mathsf{bad}] \leq \sum_{\theta}$$