Technical Summary: "An Improved Data Stream Summary: The Count-Min Sketch and its Applications" by Cormode and Muthukrishnan

Ravi Gaddipati, Matthew Ige, and Emily Wagner

December 14, 2016

1 Problem Statement and Overview

Consider a vector $\vec{a}(t) = [a_1(t), \ldots, a_i(t), \ldots a_n(t)]$ which evolves with time. Initally, $\vec{a}$ is the zero vector $\vec{0}$. We represent the tth update as (i_t, c_t), which modifies the vector as follows:

$$a_{i_t}(t) = a_{i_t}(t-1) + c_t, \tag{1}$$
$$a_{i'}(t) = a_{i'}(t-1),\ i' \neq i_t \tag{2}$$

The update (i_t, c_t) modifies the ith element by adding c_t to it. For all other $a_{i'}$, the vector remains unchanged.

This vector and its updates represent some stream of data that evolves with time. There are 2 main models for such a stream:

1. **cash-register case:** $c_t > 0$, so every vector element is monotonically increasing.
2. **turnstile case:** c_t can also be negative. There are two subcases:
   (a) **non-negative turnstile:** (i_t, c_t) will never cause vector elements a_{i_t} to dip below zero. This guarantee can be provided by the application.
   (b) **general turnstile:** vector elements a_{i_t} may become negative.

The basic problem this work addresses is to summarize or calculate certain characteristics about the stream. There are 2 main constraints. First, the space used by such algorithms should be small; at most polylogarithmic in n. This compressed version of the data is called a **sketch**. Second, updates to the sketch should be processed quickly.

Since we are sublinear in input space, we will have to approximate almost any function we want to compute over $\vec{a}$, but we still want to specify an approximation parameter ε and bound the probability of error by δ.

The **count-min sketch** addresses these requirements and allows us to calculate several characteristics of a stream using a single type of data structure. It can be used to approximate three types of **queries**:

1. **point query:** $Q(i)$ is an approximation of the vector element $a_i(t)$
2. **inner product query:** $Q(\vec{a}, \vec{b})$ approximates $\vec{a} \odot \vec{b} = \sum_{i=1}^{n} a_i b_i$
3. **range query:** $Q(l, r)$ is an approximation of $\sum_{i=l}^{r} a_i$.

These queries can be used for more complex data stream functions, such as ϕ-quantiles and heavy hitters, which will be described in section 7.

2 State of the art

Sketches have been commonly used to help analyze large streams of data. Although these data structures have proved to be very powerful, there are a few key drawbacks that limit the effectiveness of these sketches:

1. Common sketches ([4], [3], [1]) typically use $\Omega(\frac{1}{\varepsilon^2})$ space. For common uses of ε, like 0.1 or 0.01, this scaling can quickly become too expensive.
2. Many sketches take linear time (in the size of the sketch) to do a single update([4], [1]).
3. Some sketches require 4-wise independent hash functions [4]. This is not trivial, and is especially difficult in hardware applications.
4. Some sketches can only handle one particular query.
5. Many sketches use analysis that hides large constants ([3]).

The count-min sketch reduces these problems in the following ways:

1. CMS uses space proportional to $\frac{1}{\varepsilon}$.
2. Update time is sublinear with respect to the size of the sketch.
3. CMS requires only pairwise independent hash functions.
4. The sketch can answer several queries and has numerous applications.
5. All constants are explicit and small.

In particular, notice that the space has been reduced from $\frac{1}{\varepsilon^2}$ to $\frac{1}{\varepsilon}$ using a CMS, and the time for an update has been reduced from $\frac{1}{\varepsilon^2} \ln \frac{1}{\delta}$ to $\ln \frac{1}{\delta}$ (count sketch also has this update time, however).

3 How to make a count-min sketch

Note e is the base of the natural logarithm function, ln. The count-min sketch is an array with w columns and d rows where:

$$w = \lceil e/\varepsilon \rceil \tag{3}$$

$$d = \lceil \ln 1/\delta \rceil \tag{4}$$

for a given accuracy parameter ε and probability guarantee δ. We need d pair-wise independent hash functions

$$h_1 \dots h_d : \{1 \dots n\} \rightarrow \{1 \dots w\} \tag{5}$$

The sketch starts with every array entry being 0. The count-min sketch *count* is updated when the tth update $\{a_{i_t}, c_t\}$ arrives as follows

$$count[j, h_j(i_t)] \leftarrow count[j, h_j(i_t)] + c_t \tag{6}$$

In other words, upon receiving an update (i_t, c_t):

```
Update_CM_Sketch(count, (i_t, c_t))
    for each row j, 1 ≤ j ≤ d:
        count[j, h_j(i)] ← count[j, h_j(i)] + c_t
    return count
```

4 Approximate Point Queries using CM sketches

4.1 Non-Negative Case

4.1.1 Algorithm

For the non-negative case (either the non-negative turnstile or cash-register case), the algorithm to estimate $\hat{a}_i(t) = Q(i)$ is as follows:

$$Q(i) = \hat{a}_i = \min_j count[j, h_j(i)] \tag{7}$$

In other words for a given $\hat{a}_i(t) =$ `Estimate_Point_Query(count, i_t)`.

```
Estimate_Point_Query(count, i_t)
    s ← ∞
    for each row j, 1 ≤ j ≤ d:
        if count[j, h_j(i)] < s:
            s ← count[j, h_j(i)]
    return s;
```

Theorem 1: The estimate $\hat{a}_i$ has the following guarantees:

(1) $a_i \leq \hat{a}_i$

(2) with probability at least $1 - \delta$, $\hat{a}_i \leq a_i + \varepsilon ||\vec{a}||_1$

4.1.2 Proof of Theorem 1

First, we prove 1.a.

Define an indicator variable $I_{i,j,k}$ as follows:

$$I_{i,j,k} = \begin{cases} 1, & \text{if } h_j(i) = h_j(k) \text{ and } i \neq k \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

$I_{i,j,k}$ is 1 if i and k, $i \neq k$, have a collision under hash function h_j.

Since, to begin with, we chose all h_j from a 2-universal hash family with range $1 \leq h_j \leq d = \lceil \frac{e}{\varepsilon} \rceil$ we know that, for $i \neq k$,

$$\begin{aligned} Pr(h_j(i) = h_j(k)) &\leq \frac{1}{\text{range}(h_j)} = 1/\lceil \frac{e}{\varepsilon} \rceil = \lfloor \frac{\varepsilon}{e} \rfloor \\ \implies E(I_{i,j,k}) &\leq \frac{\varepsilon}{e} \end{aligned} \tag{9}$$

by the properties of a 2-universal hash function and by our choice of hash function range.

Define $X_{i,j}$ as follows:

$$X_{i,j} = \sum_{k=1}^{n} I_{i,j,k} a_k \tag{10}$$

In other words, $X_{i,j}$ is the sum of all corresponding vector values whose indices i collide under h_j. Trivially, we can then state that

$$count[j, h_j(i)] = a_i + X_{i,j} \tag{11}$$

Now, Theorem 1.a follows because we are only considering the case where vector elements are non-negative and thus

$$a_i \leq \hat{a}_i = a_i + X_{i,j} \tag{12}$$

Now we show 1.b.

For future use in the analysis, we evaluate:

$$E(X_{i,j}) = E\left(\sum_{k=1}^{n} I_{i,j,k} a_k \right) \tag{13}$$

by definition of $X_{i,j}$. The linearity of expectation states that

$$E(\sum_{i=1}^{n} c_i X_i) = \sum_{i=1}^{n} c_i E(X_i) \tag{14}$$

where X_is may be dependent. Applying linearity of expectations here, we have:

$$E(\sum_{k=1}^{n} a_k I_{i,j,k}) = \sum_{k=1}^{n} a_k E(I_{i,j,k}) \tag{15}$$

We showed $E(I_{i,j,k}) \leq \frac{\varepsilon}{e}$ in Equation 9, so we have

$$\sum_{k=1}^{n} a_k E(I_{i,j,k}) \leq \sum_{k=1}^{n} a_k \frac{\varepsilon}{e} \tag{16}$$

and finally, since $||\vec{a}||_1 = \sum_{k=1}^{n} |a_k| = \sum_{k=1}^{n} a_k$ (all a_k are assumed to be non-negative)

$$\begin{aligned} \sum_{k=1}^{n} a_k \frac{\varepsilon}{e} &= \frac{\varepsilon}{e}||\vec{a}||_1 \\ \implies E(X_{i,j}) &\leq \frac{\varepsilon}{e}||\vec{a}||_1 \end{aligned} \tag{17}$$

We will use this result in the following proof.

We want to show that $Pr[\hat{a}_i > a_i + \varepsilon||a||_1] < \delta$. We can see that

$$Pr[\hat{a}_i > a_i + \varepsilon||a||_1] = Pr[\forall_j.count[j, h_j(i)] > a_i + \varepsilon||\vec{a}||_1] \tag{18}$$

since the probability that all values in a set are greater than $a_i + \varepsilon||\vec{a}||_1$ is equivalent to the probability that the minimum of all the values in that set is greater than $a_i + \varepsilon||\vec{a}||_1$. (Recall that $\hat{a}_i$ is defined as the minimum of all of the array elements that i hashes to in each row.) Then, by the definition stated in equation 11, we have

$$Pr[\forall_j.count[j, h_j(i)] > a_i + \varepsilon||\vec{a}||_1] = Pr[\forall_j.a_i + X_{i,j} > a_i + \varepsilon||\vec{a}||_1] \tag{19}$$

Then, subtracting a_i from both sides, and using equation 17 on the $\varepsilon||\vec{a}||_1$ term, we have:

$$Pr[\forall_j.a_i + X_{i,j} > a_i + \varepsilon||\vec{a}||_1] \geq Pr[\forall_j.X_{i,j} > eE(X_{i,j})] \tag{20}$$

Finally, consider the Markov inequality:

$$Pr[X > aE[X]] \leq \frac{1}{a} \tag{21}$$

Since this is the non-negative case and thus $X_{i,j}$ is non-negative, we can apply it here:

$$Pr[X_{i,j} > eE(X_{i,j})] \leq \frac{1}{e} \tag{22}$$

$$\implies \bigcap_{1 \leq j \leq d} Pr[X_{i,j} > eE(X_{i,j})] \leq \left(\frac{1}{e}\right)^d \tag{23}$$

$$\implies Pr[\forall_j.X_{i,j} > eE(X_{i,j})] \leq e^{-d} \leq \delta \tag{24}$$

Note that we can use the intersection bound here because our hash functions for each row were picked independently and uniformly at random from a 2-universal hash-family.

This estimate is calculated in $O(\ln \frac{1}{\delta})$ as the minimum of a multiset can be taken in linear time, and we are taking it across $d = O(\ln \frac{1}{\delta})$ rows for the point query. The update time to maintain the data structure after each update is also $O(\ln \frac{1}{\delta})$, since we hash a single value in each row of the data structure.

The space complexity is $O\left(\frac{e}{\varepsilon} \ln \frac{1}{\delta}\right)$ words, as the dimensions of the matrix are $w \times d = \lceil \frac{e}{\varepsilon} \rceil \times \lceil \ln \frac{1}{\delta} \rceil$. $O(\ln \frac{1}{\delta})$ is also required to store the hash functions. The constants are clearly small, since this is the total space requirement.

All previous analyses of sketch algorithms use Chebyshev in their estimation analysis, yielding a dependency on $\frac{1}{\varepsilon^2}$ for the space complexity. Using Markov in this analysis yields a tighter bound, with a dependency $\frac{1}{\varepsilon}$.

4.2 General Case

4.2.1 Algorithm

The algorithm is identical to the non-negative case, except that we take the median among the rows:

$$Q(i) = \hat{a}_i = \text{median}_j count[j, h_j(i)] \tag{25}$$

Theorem 2: With probability $1 - \delta^{1/4}$,

$$a_i - 3\varepsilon ||\vec{a}||_1 \leq \hat{a}_i \leq a_i + 3\varepsilon ||\vec{a}||_1. \tag{26}$$

Here, since the array elements can be negative, we can no longer use Markov, and instead must apply Chernoff bounds.

The time and space complexity are the same as the non-negative case: $O(\ln \frac{1}{\delta})$ time and $O(\frac{e}{\varepsilon} \ln \frac{1}{\delta})$ words, respectively.

5 Inner Product Query

5.1 Algorithm: Non-negative case

Two CM sketches are kept, one for vector a and one for vector b. Updates to each sketch take the form $(i_t, c_t)_v : count_v(j, h_j[i]) \leftarrow count_v(j, h_j[i]) + c_t$. To compute the estimated inner product $\widehat{\vec{a} \odot \vec{b}}$, we compute the inner product of each row pair $count_a[j] \cdot count_b[j]$ for all $j \in d$, taking the minimum. Formally, the estimate for the inner product $Q(\vec{a}, \vec{b})$ for non-negative vectors $\vec{a}$ and $\vec{b}$ is $\widehat{\vec{a} \odot \vec{b}} = \min_j (\widehat{\vec{a} \cdot \vec{b}})_j$, where $(\widehat{\vec{a} \cdot \vec{b}})_j = \sum_{k=1}^{w} count_a[j,k] * count_b[j,k]$. In psuedocode form:

```
Estimate_Inner_Product(count_a, count_b):
    make empty set Estimates
    for each row j, 1 ≤ j ≤ d:
        Estimates[j] ← Σ_{k=1}^{w} count_a[j,k] * count_b[j,k]
    return min(Estimates)
```

Theorem 3:
The estimate $\widehat{\vec{a} \odot \vec{b}}$ has the following guarantees:

(1) $\vec{a} \odot \vec{b} \leq \widehat{\vec{a} \odot \vec{b}}$

(2) with probability at least $1 - \delta$,

$$Pr(\widehat{\vec{a} \odot \vec{b}} \leq \vec{a} \odot \vec{b} + \varepsilon ||\vec{a}||_1 ||\vec{b}||_1) \tag{27}$$

5.2 Proof of Theorem 3

We represent the estimated inner product as the sum of the true inner product and collisions. By the pairwise independence of all hash functions h_j, $1 \leq j \leq d$,

$$(\widehat{\vec{a} \cdot \vec{b}})_j = \sum_{i=1}^{n} a_i b_i + \sum_{p \neq q,\ h_j(p) = h_j(q)} a_p b_q \tag{28}$$

Taking the expected value of the error term shows that the error is dependent on the probability of collision.

$$E\left[(\widehat{\vec{a} \cdot \vec{b}})_j - \vec{a} \cdot \vec{b}\right] = E\left[\sum_{p \neq q,\ h_j(p) = h_j(q)} a_p b_q\right] \tag{29}$$

$$= \sum_{p \neq q} Pr(h_j(p) = h_j(q)) a_p b_q \tag{30}$$

Using the results from Theorem 1, we know the probability of collision is bounded by ε/e where ε is the parameter used during sketch construction.

$$\leq \sum_{p \neq q} \frac{\varepsilon}{\mathrm{e}} a_p b_q \tag{31}$$

$$E\left[(\widehat{\vec{a} \cdot \vec{b}})_j - \vec{a} \cdot \vec{b}\right] \leq \frac{\varepsilon}{\mathrm{e}} ||a||_1 ||b||_1 \tag{32}$$

This follows by the treatment of the inner product computation as a series of point queries. Applying the Markov inequality we bound the probability of the error.

$$Pr((\widehat{a \cdot b})_j - \vec{a} \cdot \vec{b} > \varepsilon ||a||_1 ||b||_1) \leq \frac{E[\widehat{a \cdot b} - a \cdot b]}{\varepsilon ||a||_1 ||b||_1} \tag{33}$$

$$\leq \frac{\frac{\varepsilon}{\mathrm{e}} ||a||_1 ||b||_1}{\varepsilon ||a||_1 ||b||_1} \tag{34}$$

$$\leq 1/e \tag{35}$$

As defined in the construction of the CM sketch, $d = \ln 1/\delta$, so $\delta = e^{-d}$. The probability the error exceeds this threshold for a single estimator is less than or equal to $1/e$ as shown above. Since we take the minimum inner product as our estimate, we compute the probability all estimators exceed this threshold.

$$\bigcap_{1 \leq j \leq d} Pr(\widehat{(\vec{a} \cdot \vec{b})}_j - \vec{a} \cdot \vec{b} > \varepsilon ||a||_1 ||b||_1) \leq \left(\frac{1}{\mathrm{e}}\right)^d \tag{36}$$

$$\leq \delta \tag{37}$$

for all $d > 0$.

6 Range Query: Non-negative

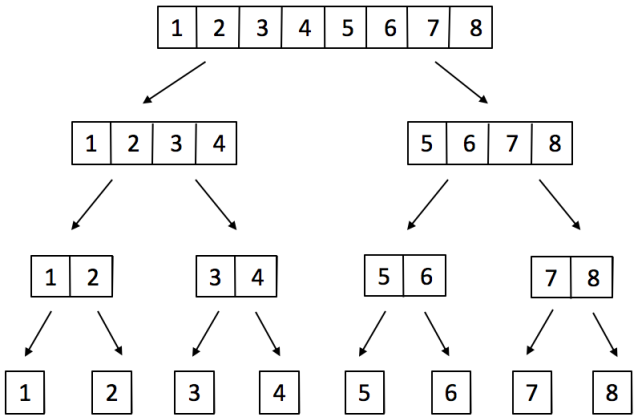
A range query, denoted as: $\mathcal{Q}(l, r)$, requests the value:

$$a[l, r] = \sum_{i=l}^{r} a_i$$

And we denote our estimator based on the CM sketch as: $\hat{a}[l, r]$. It is possible to simply query all values in the range, $\hat{a}[l, r] = \sum_{i=l}^{r} \hat{a}_i$, however, this query would take linear time with respect to the input. Additionally, our error bound would have an extra multiplicative term in the length of the range, $r - l$. We can utilize dyadic ranges to help address both of these issues.

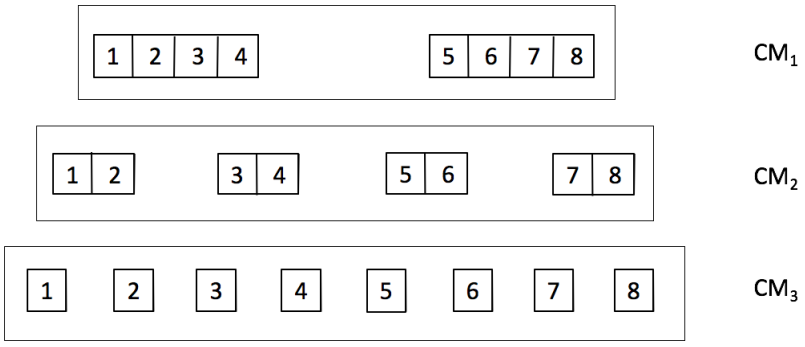
6.1 Dyadic ranges

For any range, we can create a set of dyadic ranges by splitting the range in half to get two ranges of half the size. We can continually do this, until we have ranges of size 1. We can think of the construction of a set of dyadic ranges as creating a tree-like structure:

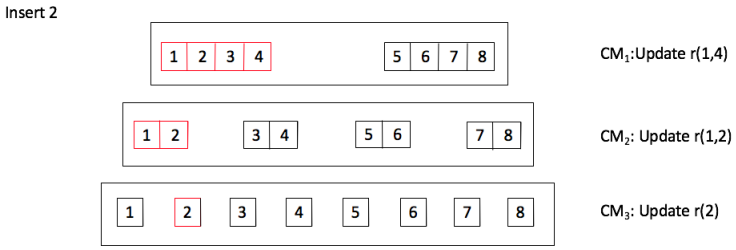


6.2 Range Query: updates

The update procedure is as follows: Keep $\log_2 n$ CM sketches. Each CM sketch represents one level of our "tree" of dyadic ranges, and the keys for that CM sketch are the ranges within that level, and the values are the sum of the counts for all values in the range of each key:



When we get an update, (i_t, c_t), we simply update each CM sketch, with the key as the range that covers the value i_t:

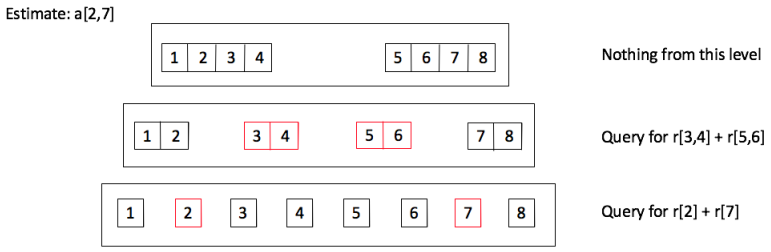


We thus have the following pseudocode for an update, where `counts` is all of the CM sketches in the tree, and $range_i(i_t)$ represents the dyadic range that contains i_t, for level i:

```
Range_Query_Update(counts, (i_t, c_t))
    for each sketch, count_i ∈ counts:
        Update_CM_Sketch(count_i, (range_i(i_t), c_t))
    return counts;
```

6.3 Range Query: estimation

Then, in order to estimate a range query, $\hat{a}[l, r]$, we identify which dyadic ranges contribute to our overall range, and query those in each appropriate CM sketch.



A single range can be converted into at most $2\log_2 n$ dyadic ranges, because in the worst case, both sides of the range query (the upper bound r and lower bound l) must fully travel through the height of the tree. Because we store each dyadic range as a key value in a CM sketch, we do a total of, at most, $2\log_2 n$ point queries. We then sum together the point queries, to get our total range query estimate. In pseudocode form, the procedure for estimation is as follows:

```
Estimate_Range_Query(counts, a[l,r], i, x, y)
    if range(x,y) ∈ [l,r]:
        return Estimate_Point_Query(count_i, range(x,y))
    else:
        return Estimate_Range_Query(counts, a[l,r], i+1, x, (x+y)/2) +
            Estimate_Range_Query(counts, a[l,r], i+1, (x+y)/2 + 1, y)
```

Where i denotes the level that we are looking at in our tree of dyadic ranges, $count_i$ represents the CM sketch of that level, and each recursive call simply alters the range we are looking for, along with going down one level in our tree.

Finally, recall the idea behind using dyadic ranges: to reduce the estimation time to be poly-logarithmic in the size of the input. A single point query update/estimation takes $\log(\frac{1}{\delta})$ time. In order to do a single update, we update $O(\log n)$ CM sketches. When we estimate our range query, we do $O(\log n)$ point query estimations. Thus, for both an update and an estimation, our time is reduced to $O(\log(n)\log(\frac{1}{\delta}))$.

6.4 Theorem 4:

(1) $a[l,r] \leq \hat{a}[l,r]$.

(2) With probability at least $(1-\delta)$, $\hat{a}[l,r] \leq a[l,r] + 2\varepsilon \log n ||\vec{a}||_1$

Proof:
Our procedure for estimation covers exactly the range [l,r]. We are summing up multiple point queries, and thus, based on the previous proof, (1) holds. In order to show (2), the following

notation will be used:

$$\begin{aligned} range[l,r] &= \text{The range from l through r} \\ range_i &= \text{A dyadic range.} \\ \bigcup_{range_i \in range[l,r]} range_i &= \text{A set of non-overlapping dyadic ranges that exactly cover range[l,r]} \end{aligned}$$

Thus, we show:

$$\begin{aligned} \hat{a}[l,r] &= \sum_{range_i \in range[l,r]} a_{range_i} + \sum_{i \neq k, h_j(range_i) = h_j(range_k)} a_{range_i} \\ E[\hat{a}_j[l,r] - a[l,r]] &= E[\sum_{range_i \in range[l,r]} a_{range_i} + \sum_{i \neq k, h_j(range_i) = h_j(range_k)} a_{range_i} - \sum_{range_i \in range[l,r]} a_{range_i}] \\ &= E[\sum_{i \neq k, h_j(i) = h_j(k)} a_{range_i}] \\ &\leq 2\frac{\varepsilon}{\mathrm{e}} \log n ||a||_1 \end{aligned}$$

Where the error for a single point query is $\frac{\varepsilon}{\mathrm{e}}||a||_1$, and we do at most $2log_2 n$ point queries. Thus, the error is bounded as above. We can then use the Markov inequality to bound the error probability:

$$\begin{aligned} p(\hat{a}_j[l,r] - a[l,r] > 2\varepsilon \log n ||a||_1) &\leq \frac{E[\hat{a}_j[l,r] - a[l,r]]}{2\varepsilon \log n ||a||_1} \\ &\leq \frac{2\frac{\varepsilon}{\mathrm{e}} \log n ||a||_1}{2\varepsilon \log n ||a||_1} \\ &\leq \frac{1}{\mathrm{e}} \end{aligned}$$

However, this is the error bound for a single estimator. Our CM sketch construction uses d rows, and thus, to bound the error, we need to find the probability that all of our estimators exceed the threshold:

$$\begin{aligned} \bigcap_{1 \leq j \leq d} Pr(\hat{a}_j[l,r] - a[l,r] > 2\varepsilon \lg n ||\vec{a}||_1) &\leq \left(\frac{1}{\mathrm{e}}\right)^d = (\mathrm{e})^{-d} \\ &\leq \delta \end{aligned}$$

Therefore, with probability at least $(1-\delta)$, $\hat{a}[l,r] \leq a[l,r] + 2\varepsilon \log n ||\vec{a}||_1$

7 Applications

7.1 Quantiles in the turnstile model

Define the ϕ-quantiles of the cardinality

$$||\vec{a}||_1 = \sum_{i=1}^{n} |a_i(t)| \tag{38}$$

as the values within a multiset of finite size that split it into ϕ groups of equal size based on their rank R. For example, the median of a set of data is the 2-quantile.

To obtain the ϕ-quantiles, the elements are sorted and split as follows. The kth quantile is the element(s) with rank $R = k\phi||\vec{a}||_1$ for $k = 1, 2, \ldots, 1/\phi$. The ε-approximation for the ϕ-quantiles accepts as the k-th quantile any of the values with rank $(k\phi - \varepsilon)||\vec{a}||_1 \leq R \leq (k\phi + \varepsilon)||\vec{a}||_1$ for a given $\varepsilon < \phi$.

The count-min sketch can be used to calculate the ϕ-quantiles in the turnstile model. In this case, updates where $c_t < 0$ are called deletions, and $c_t > 0$ are called insertions. The value for each vector element a_i is actually the number of instances of the element i.

Prior work [1] shows that ϕ-quantiles in this model can be approximated using range sums. This is done as follows:

1. For each $k \in 1, 2, \ldots, 1/\phi$
   (a) Find a range sum such that $a[1, r]$ is the ε-approximation for $k\phi||\vec{a}||_1$. Find r using a binary search on the possible range sums $a[1, \hat{r}]$, where $\hat{r} \in \{1, 2, \ldots, n\}$. r is the ε-approximation for the kth ϕ-quantile.

The method in [1] uses Random Subset Sums to approximate range sums. If instead a count-min sketch is used to approximate the range sums, better results follow.

To use a count-min sketch to approximate quantiles in the turnstile model, $\log n$ sketches are kept, one for each dyadic range. Each sketch gets accuracy parameter $\varepsilon/\log n$ for an overall accuracy bounded by ε. Each sketch gets probability guarantee $\delta\phi/\log n$, for an overall probability guarantee of δ for all $1/\phi$ quantiles. Then, we have the following:

Theorem 5: ε-approximate ϕ-quantiles can be found with probability at least $1 - \delta$ by keeping a data structure with space $O\left(\frac{1}{\varepsilon}\log^2(n)\log\left(\frac{\log n}{\phi\delta}\right)\right)$ The time for each insert or delete operation is $O\left(\log(n)\log\left(\frac{\log n}{\phi\delta}\right)\right)$, and the time to find each quantile on demand is $O\left(\log(n)\log\left(\frac{\log n}{\phi\delta}\right)\right)$

This improves the existing query time and update by a factor of more than $\frac{34}{\varepsilon^2}\log n$ The space requirements are improved by a factor of at least $\frac{34}{\varepsilon}$.

7.2 Heavy Hitters

A potential application of the count min sketch is the heavy hitters problem. Given an input A of length n and a parameter k we wish to return values that occur in the input at least n/k times.

There may be at most k such heavy hitters, though there also may be none. This sketch provides a convenient way to find the heavy hitters of some data stream with a large input size for which maintaining a counter for each unique element is not feasible. We look at two cases:

1. The cash register case: elements are only added
2. The turnstile case: elements can be added or removed

7.2.1 Cash Register Case

Algorithm:
We can easily obtain $||\vec{a}||_1$ at any given point in time because it is simply: $\Sigma_{i=1}^{t} c_i$.We define a ϕ heavy hitter if the estimation of the point query, $Q(i_t) = \hat{a}_{it} \geq \phi||\vec{a}(t))||_1$. We can do this by maintaining a heap to store the items above the $\phi||\vec{a}(t)||_1$ threshold. On any update, we check the lowest value in the heap, and if the update would be greater than the lowest item, we replace it in the heap. When we are finished with the stream, we do a final scan of all items in the heap, and return the ones which have a value over $\phi||\vec{a}||_1$.

Theorem 6: We can identify the heavy hitters of a sequence of length $||\vec{a}||_1$ using space $O(\frac{1}{\varepsilon} log(\frac{||\vec{a}||}{\delta}))$ and time $O(log(\frac{||\vec{a}||}{\delta}))$. Every item which occurs with count at least $\phi||\vec{a}||_1$ is output, and with probability $(1-\delta)$, no items whose count is less than $(\phi-\varepsilon)||\vec{a}||_1$ are output.

Proof idea:
Because we have only positive updates, and a CM sketch will only over estimate any a_i, it is not possible to miss any heavy hitter. In order to calculate the error bound, we simply scale the parameter, δ, such that the probability of erroring at any point throughout the stream is bounded by $(1-\delta)$. This requires using the union bound for time t, and solving for the appropriate δ.

7.2.2 Turnstile Case

Algorithm:
The method outlined by Graham Cormode et. al. [2] is adapted using the count min sketch. In this scheme, a sketch is maintained for each of the $\log n$ dyadic ranges. These sketches are kept in a binary tree. When an update (i_t, c_t) is received, a tree search for a_i is performed. As the tree is traversed, the CM sketch at each level is updated to reflect (i_t, c_t). To find the heavy hitters, a binary search is performed recursively searching each range for which the counter weight exceeds $(\phi+\varepsilon)||a||_1$.

Theorem 7:
The algorithm uses space $O((1/\varepsilon)\log n \log(2\log n/(\delta\phi)))$ and time $O(\log n \times \log(2\log n/(\delta\phi)))$ per update. Every item with frequency at least $(\phi+\varepsilon)||a||_1$ is reported, and with probability $1-\delta$ no item whose frequency is less than $\phi||a||_1$ is output.
Proof idea:
Since at each level there are at most $1/\phi$ items with frequency greater than ϕ, the probability of failure for each sketch is set at $\delta\phi/(2\log n)$ to limit the number of output items whose true frequency is less than ϕ. By doing this the total probability that there is any overestimated by more than

ε is bounded by δ by the union bound. Since the sketch at minimum reports the true count, we are guaranteed that the the true heavy hitters, if they exist, are reported. With probability δ, the algorithm will report a false heavy hitter.

References

[1] S. Muthukrishnan-M. Straus A.C. Gilbert, Y. Kotidis. How to summarize the universe: dynamic maintenance of quantiles. In *28th International Conference on Very Large Data Bases*, pages 454–465, 2002.

[2] D. Srivastava G. Cormode, S. Muthukrishnan. Finding heirarchical heavy hitters in data streams. In *International Conference on Very Large Data Bases*, pages 296–306, 2003.

[3] M. Farach-Colton M. Charikar, K. Chen. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703, 2002.

[4] Y. Matias-M. Szegedy N. Alon, P. Gibbons. Tracking join and self-join sizes in limited storage. In *Eighteenth ACM Symposium on Principles of Database Systems*, pages 10–20, 1999.