Finding frequent items over general update streams

Sumit Ganguly, Abhayendra N. Singh and Satyam Shankar

IIT Kanpur

Abstract. We present novel space and time-efficient algorithms for finding frequent items over general update streams. Our algorithms are based on a novel adaptation of the popular dyadic intervals method for finding frequent items. The algorithms improve upon existing algorithms in both theory and practice.

1 Introduction

There is a growing class of applications in areas of business and scientific data processing that continuously monitor large volumes of rapidly arriving data for detecting user-programmed scenarios, some of which may encode anomaly and exception conditions or desirable conditions. Although a deep analysis of the data can be done, it is both space and time consuming. Data streaming systems are designed to give fast, but possibly approximate answers to a class of queries while processing the input data in an online fashion. For example, consider a satellite data processing system where continuous and voluminous weather data has to be rapidly processed to give a forewarning of an emerging climate phenomenon. While deep analysis is possible, often, an early warning capability is very desirable, which though approximate, could then be used to trigger a deeper analysis. As another example, consider a biological experiment scenario where there are sensors attached to many biological subjects whose data is being continuously transmitted to a central server. Monitoring extremal aggregate conditions over to sensor readings are often useful indicators in such scenarios.

Central to the success of data streaming systems are highly space and time-efficient algorithms that can summarize input data streams while processing them in an online fashion. In this paper, we present novel algorithms for data stream processing in the same vein, specifically considering general data streams. In the general stream model, each input record indicates arbitrary insertions or deletions of an item, where, an item may be an IP-address, stock ticker, sensor-id, etc.. In this model, the sum of aggregate insertions (positive) and deletions (negative) for each item over the course of the stream may be either positive or negative. We address the problem of finding frequent items over general data streams.

The problems of finding frequent items and estimating item frequencies over data streams are among the most popular primitive operations over data streams [2, 4, 3, 5, 8, 10]. Much of the research in this basic problem has centered around

the insert-only streaming model [2, 5, 8, 10] and the strict update models [3, 6] respectively. For general streams, there are two known approaches towards the problem of finding approximate frequent items, namely, the non-adaptive group testing approach [4] and the reversible sketches approach [11]. In this paper, we present the random dyadic approach towards finding frequent items over general streams. The proposed algorithm is novel, and extends the applicability of the popular dyadic intervals technique for strict streams to general streams.

Data Streaming Model. A data stream σ over the domain $[1, n] = \{1, 2, \ldots, n\}$ is modeled as an unbounded sequence of records of the form $(pos, i, \delta v)$, where, pos is the current sequence index, $i \in [1, n]$ and $\delta v \in \mathbb{Z}$. Here, $\delta v > 0$ signifies insertion(s) of instance(s) of i and $\delta v < 0$ signifies deletion(s) of instance(s) of i. For each data item $i \in [1, n]$, its frequency $f_i(\sigma)$ is defined as

$$f_i(\sigma) = \sum_{(pos,i,\delta v)\ \in\ \text{stream}} \delta v, \quad i \in [1, n]\ .$$

In this paper, we consider the *general model*, where, the n-dimensional frequency vector $f(\sigma) \in \mathbb{Z}^n$. The frequency moment F_1 of a general stream is defined as the sum of the absolute values of the frequencies, that is, $F_1(\sigma) = \sum_{i\in[1,n]}|f_i(\sigma)|$. The second moment of the frequency vector is defined as $F_2(\sigma) = \sum_{i\in[1,n]}(f_i(\sigma))^2$. The data stream model of processing permits online computations over the input sequence using sub-linear space.

Conventions. (a) We will assume that the domain size n is a power of two. (b) By a data stream, we always mean the current state of the stream and hence we drop the stream argument σ; for example, f_i abbreviates $f_i(\sigma)$.

Problem definitions. In this paper, we consider the following two problems. Let $0 < \phi < \epsilon < 1$.

1. *Finding F_1-based frequent items*, denoted by $\textsc{ApproxFreq}_1(\epsilon, \phi)$ is: return all $i \in [1, n]$ such that $f_i(\sigma) \geq \epsilon F_1$ and do not return any i such that $f_i \leq (\epsilon - \phi)F_1$. A randomized algorithm for this problem satisfies the above property for all items returned with probability $1 - \delta$.
2. *Finding F_2-based frequent items*, denoted by $\textsc{ApproxFreq}_2(\epsilon, \phi)$ is: return all items $i \in [1, n]$ such that $|f_i| \geq (\epsilon F_2)^{1/2}$, and no i such that $|f_i| < ((\epsilon - \phi)F_2)^{1/2}$. A randomized algorithm satisfies the above properties with a total success probability of at least $1 - \delta$.

In this paper, we design randomized algorithms for finding F_1 and F_2-based frequent items whose space requirement is nearly linear in ϕ^{-1}.

Contributions. We present novel, space and time-efficient algorithms to solve the problems stated above. For the problem of finding frequent items, our technique extends the applicability of the popular dyadic intervals technique for strict streams to general streams. We present two algorithms for the problem

APPROXFREQ$_2(\epsilon, \phi)$ which improve the space requirement of the existing algorithm [4] by a factor of $O(\frac{1}{\phi})$. The solution to the F_1-based frequent items problem is shown to have better properties of precision and recall. The algorithms perform well in experiments and have rigorous space versus accuracy guarantees.

2 Review

In this section, we review relevant algorithmic techniques for processing general data streams.

2.1 Review: finding approximate frequent items

We review two approaches for finding approximate frequent items over general streams, namely, non-adaptive group testing [4] and reversible sketches [11].

Non-adaptive group testing. A collection of s hash tables $T_1, \ldots, T_s$ is kept, each consisting of b buckets numbered 1 to b. Associated with each hash table T_j is a pair-wise independent random hash function $h_j : [1, n] \rightarrow [1, b]$. Each bucket of a table contains a two dimensional array $U[0 \ldots 1, 1 \ldots \log n]$ of integer counters[1]. We refer to a specific entry of a bucket as $T_j[r].U[v][k]$, where, j is the table index in $[1, s]$, r is the bucket index in $[1, b]$, v is a bit value that is either 0 or 1 and k is a bit position with value from $[1, \log n]$. Corresponding to each stream record of the form (pos, x, Δ), the data structure (initialized to all zeros) is updated as follows. Let $x = x_{\log n} x_{\log n - 1} \ldots x_2 x_1$ be the binary representation of x.

$$T_j[h_j(x)].U[x_k][k] = T_j[h_j(x)].U[x_k][k] + \Delta$$
$$j = 1, \ldots, s, k = 1, \ldots, \log n \ .$$

For the problem APPROXFREQ$_1(\epsilon, \phi)$ with $(\phi < \epsilon)$, b is set to $\lceil \frac{2}{\phi} \rceil$ and s is set to $O(\log((\phi\delta)^{-1}(\log(1/\phi))))$ in order to ensure that the problem APPROXFREQ$_1(\epsilon, \phi)$ is solved with error probability at most δ. In addition, a data structure for estimating F_1 of the stream to within a constant factor (say, $(1 \pm \frac{1}{8})$) is also kept. The procedure for inference is the following. A bucket $T_j[r]$ contributes at most one element x towards a set of candidate frequent items as follows. Let $\hat{F}_1$ = (estimate of F_1) $/(1 + 1/8)$. For each $j \in [1, s]$ and $r \in [1, b]$, the procedure RETRFREQUENT(j, r) is invoked for each hash table T_j and each bucket $r \in [1, b]$ of T_j to obtain a candidate set of non-NIL elements returned from the invocation RetrFrequent (j, r). These are the candidate frequent items–their frequencies are estimated by treating the data structure as a COUNT-MIN sketch structure [3] and $(x, \hat{f}_x)$ is returned as a frequent item and its estimate provided, $\hat{f}_x \geq (\epsilon - \phi)\hat{F}_1$. The space requirement of this technique

[1] For $k \in [1 \ldots \log n]$, $r \in [1, b]$ and $j \in [1, s]$, we have $T_j[r].U[0][k] + T_j[r].U[1][k] = \sum_{h_j(x)=r} f_x$. The latter quantity is stored in another counter associated with the bucket $T_j[r]$ thus reducing the storage associated with each bucket from $2 \log n$ counters to $1 + \log n$ counters. This optimization is done in the experiments.

procedure RETRFREQUENT(j, r) // $j \in [1, s], r \in [1, b]$
Returns $x \in [1, n]$ or NIL in case of perceived ambiguity.
$x := 0$;
for $k = 1$ *to* $\log n$ {
  if $(T_j[r].U[1][k] \geq (\epsilon - \phi)\hat{F}_1)$ *and* $(T_j[r].U[0][k] \geq (\epsilon - \phi)\hat{F}_1)$
    return NIL
  else if $(T_j[r].U[1][k] \geq (\epsilon - \phi)\hat{F}_1)$ $x := x + 2^{k-1}$
  else if $(T_j[r].U[0][k] < (\epsilon - \phi)\hat{F}_1))$ *return* NIL
}

is $O(\phi^{-1}(\log n)(\log F_1)(\log((\phi\delta)^{-1} \log \phi^{-1})))$ bits. The time required to process each stream update is $O((\log((\phi\delta)^{-1} \log \phi^{-1})) \log n)$.

The group testing approach was used by [4] to present algorithms for the problem APPROXFREQ$_2(\epsilon, \phi)$, that is, retrieve all items i such that $f_i > (\epsilon F_2)^{1/2}$ and not retrieve any items i with $f_i < ((\epsilon - \phi)F_2)^{1/2}$. The data structure has the same structure as the one described above; in addition to the array U kept for each hash table bucket $T_j[r]$, this structure also keeps $\log n$ AMS sketches, that is, $T_j[r].U[v][k]$ is an AMS sketch of the sub-stream defined by the items that map to bucket r of table j and have value v in bit position k. The asymptotic space requirement is $O(\frac{1}{\phi^2}(\log n)(\log F_1)(\log((\phi\delta)^{-1} \log \phi^{-1})))$ bits [4].

Reversible sketches. The reversible sketches paper [11] keeps $s = O(\log \frac{n}{\delta})$ tables T_j, where, each table has b buckets and each bucket is simply a counter that stores the sum of the frequencies of all the items that map to that bucket. A bucket $T_j[r]$ is considered to contain a potential frequent item provided, $T_j[r] \geq (\epsilon - \phi)F_1$. The reversible sketches does not keep any additional bits in the data structure to retrieve the items. Instead, the hash function is constructed in a modular manner that allows the retrieval of the items. The main problem with the approach is that the retrieval method can be very time-consuming (as we found in our experiments), since, the number of candidate frequent items can be as large as n^α, for α ranging from 0.5 to 0.9.

2.2 Review: Use of dyadic intervals

The dyadic intervals technique is a simple building block for design of algorithms for insert-only and strict streams. We briefly review the technique and its applications. Recall that we have assumed n to be a power of 2.

A dyadic interval at level l is an interval of size 2^l from the family of intervals of the form $[i2^l + 1, (i + 1)2^l]$, for $0 \leq i \leq \frac{n}{2^l} - 1$ and $0 \leq l \leq \log n$. The set of dyadic intervals of levels 0 through $\log n$ form a complete binary tree as follows. The root of the tree is the single dyadic interval $[1, n]$ and the leaf nodes are the singleton intervals. Moreover, for $0 \leq l < \log n$, each dyadic interval at level l of the form $I = [i2^l + 1, (i + 1)2^l]$ has two children at level $l - 1$, namely, the left and the right halves of I_h. The left child of I is the interval $[i2^l + 1, (2i + 1) \cdot 2^{l-1}]$ and the right child is the interval $[(2i + 1) \cdot 2^{l-1} + 1, (i + 1)2^l]$. The frequency of

a dyadic interval I is defined as the sum of the individual frequencies of items in I, and is denoted as f_I.

The following observations can be made for strict streams (i.e, $f_i \geq 0$, for all $i \in [1, n]$). Since each level 0 item belongs to one and only one dyadic interval at a given level l, the sum of the interval frequencies at level l is the same as the sum of the item frequencies at level 0, which is F_1. That is, $F_1 = \sum\{f_I \mid I$ is a dyadic interval at level $l\}$, for each $l = 0, 1, \ldots, \log n$. If an item i is frequent (i.e., $f_i \geq \epsilon F_1$), then the dyadic interval I that contains i at any level l has frequency $f_I \geq f_i \geq \epsilon F_1$ and is therefore also frequent at level l.

Frequent items algorithm using dyadic intervals. An algorithm for solving ApproxFreq(ϵ, ϕ) is as follows. For each level $l = 0, \ldots, \lfloor \log \epsilon n \rfloor$, a data structure for estimating the frequency of a given dyadic interval (for e.g., a Count-Min sketch sketch or Countsketch) is kept. The elements at level l are the set of dyadic intervals interval I at level l and the frequency of an interval I is defined as the sum of the frequencies of the items that belong to I, that is, the leaves of the sub-tree of the dyadic binary tree rooted at I: $f_I = \sum\{f_i \mid i \in I\}$. The set of dyadic intervals at level l are identified with their starting position modulo 2^l. Corresponding to a stream update (pos, x, Δ), we propagate the update $(pos, \lfloor \frac{x}{2^l} \rfloor, \Delta)$ to the data structure at level l, for $l = 0, 1, \ldots, \lfloor \log(\epsilon n) \rfloor$. The inference procedure for finding frequent items is as follows. Start from the structure at level $l_{\max} = \lfloor \log(\epsilon n) \rfloor$ and estimate the frequencies of each of the $2^{l_{\max}}$ dyadic intervals at level l_0 using the data structure. Select those intervals whose estimated frequency is at least $(\epsilon - \frac{\phi}{2})F_1$; consider its left and right child, estimate their frequencies using the structure at the next lower level, retain only those intervals whose estimated frequency is at least $(\epsilon - \frac{\phi}{2})F_1$; this process is continued until the ground level is reached and the structure at level 0 is processed.

The main problem in applying this technique to general streams is that, since, item frequencies can be negative, a frequent item or interval at level l may be contained in an interval at level $l + 1$ that is not frequent at its level.

3 Algorithm Countsketch Dyadic

In this section, we present the algorithm Countsketch Dyadic for finding frequent items over general streams with respect to the second moment. That is, the problem ApproxFreq$_2(\epsilon, \phi)$ is to retrieve all items i such that $|f_i| \geq (\epsilon F_2)^{1/2}$ and not return any i such that $|f_i| < ((\epsilon - \phi)F_2)^{1/2}$. The solution presented improves the space requirement of the current best algorithm by a factor of $O(\frac{1}{\phi})$ while preserving time-efficiency of processing stream updates and of retrieving the frequent items.

The basic idea is to randomly re-distribute the items in the dyadic intervals using random permutations. Let π be a random permutation of $[1, n]$ that is very nearly t-wise independent ($t = 3$ will suffice). A typical way of generating π is by the use of Fiestel permutations using Luby and Rackoff's technique [9]. The advantage of using Fiestel permutations is that it is very efficiently computed

and the inverse permutation is also very efficiently computed as follows. Given a number x expressed using $2m$ bits, let L denote the top-order m bits and R denote the low order m bits; thus $x = (L, R)$. A single round Fiestel permutation is a map $\pi : (L, R) = (R, L \oplus f(R))$, where, f is a t-wise independent hash function $f : [0, 2^m - 1] \rightarrow [0, 2^m - 1]$ and $\oplus$ denotes the bit-wise exclusive or operation. The inverse of a single-round Fiestel permutation is the map $(L, R) \rightarrow (f(L) \oplus R, L)$ and is thus easily computed. Luby and Rackoff show that four rounds of Fiestel permutations suffice to generate very nearly t-wise independent permutations such that the distance between the uniform distribution over $2m$ bits and the distribution of the Luby-Rackoff permutations is at most $t^2 \cdot 2^{-m}$. We note that for $t = 3$, there are known constructions for exactly 3-wise independent permutation families. However, for $t > 3$, constructions for exact independent random permutations are not known [7].

Let $\pi_1, \ldots, \pi_s$ be very nearly 4-wise independent permutations that are obtained in the manner explained above. For each π_j and each level $l = 0, \ldots, l_{\max}$, a CountSketch structure of height ck' and width w, where, $k' = \lceil \frac{1}{\phi} \rceil$ and the parameters c, w and $l_{\max}$ will be fixed in the analysis. For each $j = 1, 2, \ldots, s$, let $\xi_{j,x} \in \{-1, +1\}$ denote a four-wise random mapping for each $x \in [1, n]$ (i.e., an AMS sketch [1]). This family is independent of the sketches used by the CountSketch structures themselves. The processing of each stream record (pos, x, v) is as follows, for each $j = 1, 2, \ldots, s$ and $l = 0, 1, \ldots, l_{\max}$, the update $(pos, \lfloor \pi_j(x)/2^l \rfloor, v \cdot \xi_{j,x})$ is propagated to the CountSketch structure at level l corresponding to permutation π_j.

The retrieval of the frequent items is done as described in Section 2.2 with minor differences. The following procedure is repeated for each permutation index $j = 1, 2, \ldots, s$. The retrieval procedure starts from level $l_{\max}$ and scans all the dyadic intervals at this level and keeps those intervals whose estimated frequency is at least the threshold $((\epsilon - \frac{\phi}{2})F_2)^{1/2}$. The children of such intervals are considered in turn–these are the candidate intervals at level $l_{\max} - 1$. Among these intervals, those whose estimated frequency crosses the threshold $((\epsilon - \frac{\phi}{2})F_2)^{1/2}$ are retained, and the rest are discarded. The process continues to the next lower level in this manner until level 0 has been processed. The candidate intervals or items at a level are are those whose absolute value of the estimated frequency crosses the threshold $((\epsilon - \frac{\phi}{2})F_2)^{1/2}$. An estimate $\hat{F}_2$ of F_2 that is correct to within a relative accuracy of $1 \pm \frac{1}{4}$ and probability $1 - \frac{\delta}{2}$ is used and can obtained using the Fast-AMS algorithm of [12] that requires space $O((\log \frac{1}{\delta})(\log F_1))$ bits and time $O(\log \frac{1}{\delta})$ for processing a stream update.

3.1 Analysis

The residual second moment [2] denoted by $F_2^{res}(k)$ is the sum of the squares of the frequencies of all items in the stream, except for the top-k frequencies in terms of absolute value. More formally, if *rank* is a permutation of the items such that $|f_{rank(j)}| \geq |f_{rank(j+1)}|$, for $1 \leq j \leq n-1$, then, $F_2^{res}(k) = \sum_{j=k+1}^{n} f_{rank(j)}^2$, defined for $k \in [0, n-1]$.

For a permutation π_j, $j \in [1,s]$, $i \in [1,n]$ and level $l \in [0, l_{\max}]$, let $g_{j,l,i}$ be the frequency of the unique dyadic interval I to which $\pi_j(i)$ maps at level l. Let $\hat{g}_{j,i,l}$ denote the estimate obtained from the CountSketch structure for the unique dyadic interval at level l containing $\pi_j(i)$ at level l. Define the event $\text{NoCollision}_l(i)$ if the dyadic interval to which $\pi_j(i)$ maps at level l does not contain any of the top-k frequencies (except perhaps itself). Define

$$\text{NoCollision}(i, l_{\max}) = \text{NoCollision}_1(i) \text{ and } \text{NoCollision}_2(i) \ldots \text{ and } \text{NoCollision}_{l_{\max}}(i) \ .$$

Lemma 1. *For $1 \le j \le s$ and $i \in [1,n]$,*

$$\mathsf{Pr}\left\{ |\hat{g}_{j,i,l} - f_i \xi_{j,i}| \le \left(\frac{32 F_2^{res}(k')}{k'} \right)^{1/2}, \forall l : 0 \le l \le l_{\max} \right\} \ge \frac{5}{8} \ .$$

Proof. Fix a permutation π_j and abbreviate it by π and the corresponding sketch family as $\{\xi_i\}_{i \in [1,n]}$. Similarly, abbreviate $g_{j,i,l}$ by $g_{i,l}$, etc.. Fix a top-k element j, $j \ne i$. Let $l \in [0, l_{\max}]$. Due to t-wise independence of π_j, $t \ge 2$, the probability that i and j map to the same dyadic interval at level l is

$$\frac{\binom{n-2}{2^l-2}}{\binom{n-1}{2^l-1}} = \frac{2^l - 1}{n-1} < \frac{2^l}{n} \ .$$

Therefore, $\mathsf{Pr}\{\text{NoCollision}_l(i)\} \ge 1 - \frac{k 2^l}{n}$, by union bound. Since, $\text{NoCollision}_l(i)$ implies $\text{NoCollision}_{l'}(i)$, for $l' < l$, $\mathsf{Pr}\{\text{NoCollision}(i, l_{\max})\} \ge 1 - \frac{k 2^{l_{\max}}}{n}$. Let $k' = 8\lceil \frac{1}{\phi} \rceil$. Fix an item i. For $j \in [1,n]$ and $j \ne i$, the indicator variable $u_{l,j}$ is defined as follows: it is 1 if j maps to the same dyadic interval at level l as i and is 0 otherwise. Thus,

$$g_{l,i} = f_i \xi_i + \sum_{j \ne i} f_j \xi_j u_{l,j} \ .$$

Assuming $\text{NoCollision}_l(i)$, we have by direct calculation

$$\mathsf{E}\left[(g_{l,i} - f_i \xi_i)^2 \right] < F_2^{res}(k') \frac{2^l}{n} \ .$$

This repeats the arguments of Alon, Matias and Szegedy [1]. By Markov's inequality,

$$\mathsf{Pr}\left\{ (g_{l,i} - f_i \xi_i)^2 < t F_2^{res}(k') \frac{2^l}{n} \right\} \ge 1 - \frac{1}{t}$$

or, equivalently,

$$|g_{l,i} - f_i \xi_i| < \left(\frac{t F_2^{res}(k') 2^l}{n} \right)^{1/2} \text{ with prob. } 1 - \frac{1}{t} \ .$$

The expression $\frac{2^l}{n}$ is largest for $l = l_{\max}$. Therefore, letting $l_{\max} = \lceil \log \frac{n}{4k't} \rceil$ ensures that $\left(\frac{tF_2^{res}(k)2^l}{n}\right)^{1/2} \leq \left(\frac{F_2^{res}(k')}{4k'}\right)^{1/2}$. Therefore, with this choice of $l_{\max}$, we have

$$|g_{l,i} - f_i\xi_i| < \left(\frac{F_2^{res}(k')}{4k'}\right)^{1/2} \text{ with prob. } 1 - \frac{1}{t} \ . \tag{1}$$

Define $F_{2,l}$ to be the sum of the squares of the frequencies of the dyadic intervals at level l. For $i \in [1, n]$ and $r \in [1, \frac{n}{2^l}]$, let $v_{l,i,r} = v_{i,r}$ denote the indicator variable that is 1 if i is mapped to the dyadic interval $[r2^l + 1, (r+1)2^l]$. Therefore,

$$F_{2,l} = \left(\sum_{r=0}^{n/2^l - 1} \sum_{i=1}^{n} f_i v_{i,r} \xi_i \right)^2 .$$

By direct calculation, $\mathsf{E}\big[F_{2,l}\big] = F_2$ and $\mathsf{Var}\big[F_{2,l}\big] \leq 5F_2$. Repeating the argument of CountSketch algorithm [2], with height $32k'$ and width w at each level,

$$|\hat{g}_{l,i} - g_{l,i}| \leq \left(\frac{F_2^{res}(32k')}{4k'}\right)^{1/2} \text{ with prob. } 1 - 2^{-\Omega(w)} \ .$$

Combining with (1), we have,

$$\mathsf{Pr}\left\{\forall l : 0 \leq l \leq l_{\max} \left(|\hat{g}_{i,l} - f_i\xi_i| \leq \left(\frac{F_2^{res}(k')}{k'}\right)^{1/2} \right) \right\} \geq 1 - l_{\max}\left(2^{-\Omega(w)} + \frac{1}{t}\right) \ .$$

Choosing $l_{\max} = \lfloor \log \frac{\phi n}{32 \log(\phi n)} \rfloor$, $t = 8l_{\max}$ and $w = O(\log \log l_{\max})$, the error probability in the above expression is $\frac{2}{8}$. Since, the probability of NoCollision$(i, l_{\max})$ is $\frac{7}{8}$, combining, we obtain the lemma. □

Theorem 1 summarizes the space, accuracy and time properties.

Theorem 1. *The algorithm* CountSketch Dyadic *with height* $ck' = 32\lceil \frac{1}{\phi} \rceil$, *width* $w = O(\log \log(\phi n))$, *maximum dyadic level* $l_{\max} = \lfloor \log \frac{\phi n}{32 \log(\phi n)} \rfloor$ *and number of permutations* $s = O(\log \frac{1}{\phi\delta})$ *solves the problem* ApproxFreq$_2(\epsilon, \phi)$ *with probability* $1 - \delta$ *with the following characteristics.*

$$\begin{aligned} \textit{Space} \quad & O\left(\frac{1}{\phi}\left(\log \frac{\phi n}{\log(\phi n)}\right)\left(\log \frac{1}{\phi\delta}\right)(\log \log(n\phi))(\log F_1)\right) \\ \textit{Update Time} \quad & O\left(\left(\log \frac{\phi n}{\log(\phi n)}\right)(\log \log n)(\log \frac{1}{\phi\delta})\right) \\ \textit{Retrieval Time} \quad & O\left(\frac{\log(\phi n)}{\phi}(\log \log(n\phi))(\log \frac{1}{\phi\delta}))\right) \ . \end{aligned}$$

□

The proposed algorithm improves the space requirement for solving the ApproxFreq$_2(\epsilon, \phi)$ problem as compared to the variational deltoids algorithm [4] by reducing the dominant term in the space complexity expression from $O(\frac{1}{\phi^2})$ to $O(\frac{1}{\phi})$.

4 Algorithm COUNTSKETCH LINEAR

An improvement of the variational deltoids algorithm of [4] for the problem APPROXFREQ$_2(\epsilon, \phi)$ that reduces the dominant term in the space complexity expression from $O(\frac{1}{\phi^2})$ to $O(\frac{1}{\phi})$ can be designed although it appears to have higher constant factors than the COUNTSKETCH DYADIC algorithm discussed above. We briefly present the design and analysis of such an algorithm which we term as COUNTSKETCH LINEAR.

The data structure consists of s tables $T_1, \ldots, T_{s_1}$, each consisting of ck' buckets, where, $k' = \lceil \frac{1}{\phi} \rceil$, where, $c = 8$ and $s_1 = O(\log \frac{k' \log(1/\delta)}{\delta})$. Each bucket $T_j[r]$ has an array of sketches $U[v][k][s_2][s_3]$, where, $v \in \{0, 1\}$ denotes a bit value, $k \in [1, \log n]$ denotes a bit position, $s_2 = O(1)$ (to be fixed later) and $s_3 = O(\log \log n)$. Corresponding to each table T_j, we keep $s_2 \cdot s_3$ independent families of AMS sketches denoted by $\xi_{x,j,u,w}$, where, $x \in [1, n]$, $j \in [1, s_1], u \in [1, s_2]$ and $w \in [1, s_3]$. Each stream update of the form (pos, x, Δ) is processed as follows. Let $x = x_{\log n} x_{\log n - 1} \ldots x_2 x_1$ denote the binary representation of x.

$$T_j[h_j(x)].U[x_k][k][u][w] = \Delta \cdot \xi_{x,j,u,w},$$
$$j \in [1, s_1], k \in [1, \log n], u \in [1, s_2], v \in [1, s_3] \ .$$

The time taken to process each stream update is therefore $O(s_1 s_2 s_3 \log n) = O((\log \frac{\log(1/\delta)}{\phi\delta})(\log n)(\log \log n))$. A set of candidate frequent items is obtained by calling procedure *Retrieve*(j, r), for $j \in [1, s_1]$ and $r \in [1, h]$ as presented in Figure 1. A second verification step is then performed wherein the frequency of each candidate frequent item x is estimated as $\hat{f}_x$ by treating the structure as a standard COUNTSKETCH structure. The pair $(x, \hat{f}_x)$ is returned provided $|\hat{f}_x| \geq ((\epsilon - \frac{\phi}{2})\hat{F}_2)^{1/2}$. An estimate $\hat{F}_2$ such that $|\hat{F}_2 - F_2| \leq \frac{F_2}{4}$ is obtained using the FAST-AMS algorithm [12] using $O(\log \frac{1}{\delta})$ hash tables, each having $O(1)$ buckets.

Analysis of COUNTSKETCH LINEAR

Lemma 2. *Suppose* $s_2 \geq \frac{40\epsilon}{\epsilon - \phi/2}$, $h = ck' \geq 8\lceil \frac{1}{\phi} \rceil$. *If* $|f_x| > (\epsilon F_2^{res}(k'))^{1/2}$, *then, for any fixed* $j \in [1, s_1]$, *the probability that procedure* Retrieve$(j, h_j(x))$ *returns* x *is at least* $\frac{5}{8}$.

Proof. Fix a table index j and let $X(v, k, w) = X_j(v, k, w) = \text{avg}_{u=1}^{s_2}(T_j[h_j(x)].U[v][k][u][w])^2$. Let

$$G_{j,k}(x) = \sum \{f_y^2 \mid h_j(y) = h_j(x) \text{ and } y_k = x_k\} \text{ and}$$
$$H_{j,k}(x) = \sum \{f_y^2 \mid h_j(y) = h_j(x) \text{ and } y_k = \bar{x_k}\} \ .$$

procedure Retrieve(j, r)
Retrieves a potential candidate frequent item from $T_j[r]$
$x := 0$;
for $k := 1$ *to* $\log n$
  $c_0 := 0$; $c_1 := 0$;
  for $w = 1$ *to* s_3 *do*
    $\bar{U}[0][k][w] := \text{avg}_{u=1}^{s_2}(T_j[r].U[0][k][u][w])^2$;

    $\bar{U}[1][k][w] := \text{avg}_{u=1}^{s_2}(T_j[r].U[1][k][u][w])^2$;

    if $(\bar{U}[0][k][w] > \bar{U}[1][k][w])$ $c_0 := c_0 + 1$;

    else if $(\bar{U}[1][k][w] > \bar{U}[0][k][w])$ $c_1 := c_1 + 1$;

  endfor
  if $(c_1 > s_3/2)$ $x := x + 2^k$ *elseif* $(c_0 < s_3/2)$ *return* NIL ;
endfor
return x;

Fig. 1. Finding frequent items: Algorithm CountSketch Linear

By arguments of [1],

$$\mathsf{E}\big[X(x_k,k,w) - X(\bar{x_k},k,w\big] = G_{j,k}(x) - H_{j,k}(x),$$
$$\mathsf{Var}\big[X(x_k,k,w) - X(\bar{x_k},k,w)\big] \le \frac{5}{s_2}(G_{j,k}(x) + H_{j,k}(x))^2$$

By Chebychev's inequality,

$$\begin{aligned}\mathsf{Pr}\{X(x_k,k,w) - X(\bar{x_k},k,w) \le 0\} &\le \frac{\mathsf{Var}\big[X(x_k,k,w) - X(\bar{x_k},k,w)\big]}{(\mathsf{E}\big[X(x_k,k,w) - X(\bar{x_k},k,w)\big])^2} \\ &\le \frac{5}{s_2}\cdot\frac{G_{j,k}(x) + H_{j,k}(x)}{G_{j,k}(x) - H_{j,k}(x)} \qquad (2)\end{aligned}$$

Define the event $\text{NoCollision}_j(x)$ as: none of the top-k' items map to the same bucket as x in table T_j (except perhaps x itself). Therefore,

$$\mathsf{Pr}\{\text{NoCollision}_j(x)\} \ge 1 - \frac{k'}{ck'} = 1 - 1/c \ .$$

We have $G_{j,k}(x) \ge f_x^2 \ge \epsilon F_2^{res}(k')$. Assuming $\text{NoCollision}_j(x)$,

$$\mathsf{E}\big[H_{j,k}(x) \mid \text{NoCollision}_j(x)\big] \le \frac{F_2^{res}(k')}{ck'}$$

and therefore by Markov's inequality,

$$\mathsf{Pr}\left\{H_{j,k}(x) \le \frac{8F_2^{res}(k')}{ck'}\big|\text{NoCollision}_j(x)\right\} \ge \frac{7}{8} \ .$$

Let $k' = \lceil \frac{1}{\phi} \rceil$ and $c = 16$. Then, $\frac{8F_2^{res}(k')}{ck'} \leq \frac{\phi F_2^{res}(k')}{2}$. Substituting in (2) and assuming $\text{NoCollision}_j(x)$,

$$\mathsf{Pr}\left\{X(x_k, k, w) - X(\bar{x_k}, k, w) \leq 0\right\} \leq \frac{5\epsilon}{s_2(\epsilon - \phi/2)} \leq \frac{1}{8}, \text{ if } s_2 \geq \frac{40\epsilon}{\epsilon - \phi/2} \ . \quad (3)$$

Note that the probability in (3) depends on (a) $\text{NoCollision}_j(x)$, which holds for all k if it holds for any one, and, (b) is derived for any $G_{j,k}(x)$ and $H_{j,k}(x)$ satisfying $G_{j,k} \geq f_x^2$ and $H_{j,k}(x) \leq \frac{F_2^{res}(k')}{k'}$. Since, this is the worst case, the property holds for all k, as stated below. Suppose $s_2 \geq \frac{40(\epsilon+\phi)}{\epsilon-\phi}$. Then,

$$\mathsf{Pr}\left\{X(x_k, k, w) - X(\bar{x_k}, k, w) > 0, \ \ \forall k \in [1, \log n] \mid \text{NoCollision}_j(x)\right\} \geq \frac{7}{8} \quad (4)$$

Let $W(x, k)$ be the number of w's in $[1, s_3]$ for which $X(x_k, k, w) > X(\bar{x_k}, k, w)$. Then, $\mathsf{E}\big[W(x, k) \mid \text{NoCollision}_j(x)\big] \geq \frac{7s_3}{8}$ and by Chernoff's bounds,

$$\mathsf{Pr}\left\{W(x, k) < \frac{s_3}{2} \mid \text{NoCollision}_j(x)\right\} < e^{-9s_3/56} < \frac{1}{8\log n}, \text{ if } s_3 \geq \frac{56}{9}\ln(8\log n) \ .$$

Combining using union bounds,

$$\mathsf{Pr}\left\{W(x, k) \geq 0.5s_3, \forall k \in [1, \log n]\right\} \geq 1 - \frac{\log n}{8\log n} = \frac{7}{8} \ . \quad (5)$$

Combining the error probability using union bound, namely, $\frac{1}{8}$ for $\text{NoCollision}(x)$, the total error probability is at most $\frac{2}{8}$. Therefore, the probability that x is retrieved as a frequent item by procedure *Retrieve*(j, r) is at least $\frac{6}{8}$. □

Note that for $\phi < \epsilon$, $1 \leq \frac{\epsilon}{\epsilon - \phi/2} \leq 2$. We therefore have the following theorem.

Theorem 2. *Suppose* $|\hat{F}_2 - F_2| \leq \frac{F_2}{4}$ *with probability* $1 - \delta/2$, $s_1 = O(\log \frac{\log(1/\phi\delta)}{\phi\delta})$, $s_2 = O(1)$, $s_3 = O(\log\log n)$ *and the height of the hash tables is* $ck' = O(\lceil \frac{1}{\phi} \rceil)$. *Then the algorithm* CountSketch Linear *can be used to solve the* ApproxFreq$_2(\epsilon, \phi)$ *problem with probability* $1 - \delta$. *The characteristics are*

$$\begin{aligned} \textit{Space} \quad & O\left(\tfrac{1}{\phi} \cdot (\log n)(\log\log n)\big(\log \tfrac{\log(1/\phi\delta)}{\phi\delta}\big)(\log F_1)\right) \\ \textit{Update Time} \quad & O\left((\log n)(\log\log n)\log \tfrac{\log(1/\phi\delta)}{\phi\delta}\right) \\ \textit{Retrieval Time} \quad & O\left(\tfrac{\textit{Space}}{\log F_1}\right) \ . \ \square \end{aligned}$$

A comparison of Theorems 1 and 2 shows that the properties of CountSketch Linear and CountSketch Dyadic are similar although CountSketch Linear has slightly worse constants. Both algorithms improve over the space requirement of $O(\frac{1}{\phi^2} \cdot$ poly-log) of the variational deltoids algorithm of [4].

5 Algorithm Count-Min Dyadic

In this section, we present an extension of the COUNT-MIN algorithm for finding F_1-based frequent items for general streams by using the dyadic intervals technique. We use s random permutations $\pi_1, \ldots, \pi_s$. Corresponding to π_j, we keep a dyadic intervals based data structure for levels 0 through $l_{\max}$ as described in Section 2.2. Corresponding to each permutation π_j and each dyadic level, we keep a COUNT-MIN sketch structure of height k' and width w, where, h and w are parameters that will be fixed later. Corresponding to a stream update (pos, x, Δ), the update $(pos, \pi_j(x), \Delta)$ is propagated to the jth dyadic intervals structure. Finally, during inference of frequent items, we use the jth dyadic based structure using the algorithm described in Section 2.2, to retrieve a set of candidate items S_j, then apply the inverse permutation π^{-1} to each candidate item to obtain $\pi^{-1}(S_j)$. This step is done for each $j = 1, 2, \ldots, s$. Finally, we return those items x that occur in at least two-thirds (or a majority) of the $\pi^{-1}(S_j)$'s and return the median estimate of its estimated frequency.

Analysis. Fix a permutation index j and abbreviate $\pi = \pi_j$. We will use the notation in the statement of Theorem 3. Let $k = \lceil \frac{1}{\epsilon} \rceil$. Here top-$k$ frequencies are determined in terms of the absolute value of f_j's. For a dyadic interval I at level l, define the random variable

$$g_I = \sum_{\pi(x) \in I} f_x \ .$$

Let $g_l(i)$ denote the frequency of the node I at level l to which the item i maps.

Lemma 3. *Let $t = 8\lceil \log(\phi n) \rceil$, $l_{\max} = \lfloor \log \frac{\phi n}{4t} \rfloor$ and $w = \log \log l_{\max}$. Then,*

$$\mathsf{Pr}\left\{ \forall l : 0 \le l \le l_{\max} \left(|\hat{g}_l(i) - f_i| \le \frac{\phi F_1}{2} \right) \right\} \ge \frac{5}{8} \ .$$

Proof. Let $g_l(i)$ denote the frequency of the dyadic interval I at level l to which the item i maps. Assume $\text{NoCollision}_l(i)$ holds. Then, $\mathsf{E}\big[|g_l(i) - f_i|\big] \le \frac{F_1(k)2^l}{n}$. By Markov's inequality,

$$\mathsf{Pr}\left\{ |g_l(i) - f_i| \le \frac{tF_1(k)2^l}{n} \right\} \le \frac{1}{t} \ .$$

Define $F_{l,1}$ as the sum of the absolute values of the frequencies of the family of dyadic intervals at level l. Then, $F_{l,1} \le F_1$. If $k' \ge 8\lceil \frac{1}{\phi} \rceil$, by COUNT-MIN structure guarantees, $|\hat{g}_l(i) - g_l(i)| \le \frac{\phi F_{l,1}}{4} \le \frac{\phi F_1}{4}$, with probability $1 - 2^{-\Omega(w)}$, for each l. By triangle inequality, and using union bound to add the error probabilities,

$$\mathsf{Pr}\left\{ \forall l : 0 \le l \le l_{\max} \big(|\hat{g}_l(i) - f_i| \le \frac{\phi F_1}{4} + \frac{tF_1 2^l}{n} \big) \right\}$$
$$\ge 1 - l_{\max} \left(2^{-\Omega(w)} + \frac{1}{t} \right) \ .$$

Substituting $t = 8\lceil\log(\phi n)\rceil$, $l_{\max} = \lfloor\log\frac{\phi n}{4t}\rfloor$ and $w = \log\log l_{\max}$, we have $\frac{l_{\max}}{t} \leq \frac{1}{8}$ and $\frac{t2^l}{n} \leq \frac{t2^{l_{\max}}}{n} \leq \frac{\phi}{4}$. □

The property of the algorithm is summarized in the following theorem.

Theorem 3. *The algorithm* Count-Min Dyadic *with height* $k' = 8\lceil\frac{1}{\phi}\rceil$*, width* $w = O(\log\log(\phi n))$*, maximum dyadic level* $l_{\max} = \lfloor\log\frac{\phi n}{32\log(\phi n)}\rfloor$ *and number of permutations* $s = O(\log\frac{1}{\phi\delta})$ *solves the problem* ApproxFreq(ϵ, ϕ) *with probability* $1-\delta$ *with the following characteristics.*

$$\begin{aligned} \textit{Space} \quad & O\left(\frac{1}{\phi}\left(\log\frac{\phi n}{\log(\phi n)}\right)\left(\log\frac{1}{\phi\delta}\right)(\log\log(n\phi))(\log F_1)\right) \\ \textit{Update Time} \quad & O\left(\left(\log\frac{\phi n}{\log(\phi n)}\right)(\log\log n)(\log\frac{1}{\phi\delta}))\right) \\ \textit{Retrieval Time} \quad & O\left(\frac{\log(\phi n)}{\phi}(\log\log(n\phi))(\log\frac{1}{\phi\delta}))\right) \quad . \end{aligned}$$

□

6 Experimental Comparison

In this section, we present an experimental comparison of our algorithms with the relevant algorithms in the literature. For the problem of finding F_1-based frequent items, we compare our Count-Min Dyadic algorithm with the reversible hash method of [11] and the absolute deltoids based group testing technique of [4]. For the problem of finding F_2-based frequent items, we compare our algorithms Countsketch Dyadic and Countsketch Linear with the variational deltoids group testing technique of [4].

Experimental testbed. Our experiments were run on Intel Pentium dual core 2.80 Ghz processor with 2Gb of main memory running Fedore Core version 6. We tested the algorithms against zipfian distributions. The algorithms under comparison were given the same space (in number of bytes) and run against the same input data. In fact, since our hash function code works for table sizes in powers of 2, we give additional advantage by rounding up the space to the nearest power of 2, for algorithms in the literature that we are comparing with.

The zipdiff(z_1, z_2) *distribution.* The input data was generated to simulate general streams, with positive and negative frequencies, as follows. Two random frequency vectors distributed as per normalized zipfian distribution *zipf* with parameters z_1 and z_2 are generated and their difference is taken. Varying z_1 and z_2 gives us the various test data. Such distributions are denoted as *zipfdiff*(z_1, z_2). Such distributions typically have a set of relatively high positive values as the top frequencies of *zipf*(z_1) and a set of relatively high (in absolute value) negative values distributed as the top frequencies of *zipf*(z_2). The item frequencies are

chosen in a manner that the top frequencies in terms of absolute value of either distributions do not conflict [2].

We compare the algorithms on the standard measures of *precision* and *recall*. Recall is the percentage of the frequent items that are detected as frequent by the algorithm; thus 1− recall is the fraction of false negatives. Precision is the fraction of frequent items among the set of frequent items; thus 1− precision is the fraction of false positives.

The reversible hash algorithm [11] performs well only for a limited range of the input when there are very few frequent items in the data. Otherwise, we found that the reversible hashing algorithm generates a very large number of false positive frequent items to the tune of about two to three orders of magnitude (or more) larger than the actual number of frequent items and then attempts to eliminate them in a verification phase. In summary, for the range of tests that we performed and report below, the time required to find frequent items by the reversible hashing method was found to be higher than the other methods by at least factors of 1000 to 10000 (order of ms versus order of minutes). We therefore do not report specific experimental observations relating to the reversible hashing method.

Experiment 1: Count-Min Dyadic *vs. Absolute deltoids.* Figure 2 presents the experimental evaluation of the Count-Min Dyadic method and the absolute deltoids method of [4]. We consider frequency distribution over items with frequency distributed as the difference of zipfian distributions zipf(z) with parameters z_1 and z_2 respectively. We report results for the following three distributions. Distribution A: *zipfdiff*(0.1,0.9), distribution B: *zipfdiff*(0.4,0.5), distirbution C: *zipfdiff*(0.3,0.7). The number of distinct items was fixed at 2.1 million items (2^{21}). The total space used by the algorithms is given in the tables. For Count-Min dyadic, either 6 or 7 tables were used for each permutation, the number of permutations was set to 1 (which was surprisingly sufficient), the height of the tables was varied from 2^{12} to 2^{14} (in powers of 2) and the number of levels was set to between 19 and 21 ($l_{\max} = 32 - \log(height) + 1$). The parameters of the absolute deltoids algorithm was set so that the total space used is no less than the Dyadic algorithm–this translates to table height ranging from 2^{11} to 2^{13} (in powers of 2) and the number of tables being set to one more than that for the instance of Count-Min Dyadic being compared with.

Results and Conclusions for Experiment 1. The precision of both algorithms is close to 100% in the sense that the items reported as frequent are truly frequent (almost always). We therefore do not report precision in the tables. The two algorithms are distinguishable by their recall; the Count-Min dyadic method is

[2] This can be done in multiple ways, namely, randomized, where, the ranking of the items in terms of each of *zipf*(z_1) and *zipf*(z_2) is randomized, leading to very low probability of conflict of the few top-k items in each distribution. We perform this in a deterministic manner, where the the ranking of the items in terms of frequencies for the first distribution *zipf*(z_1) is the standard order $1, 2, \ldots, n$ whereas, the ranking of the items for the second distribution is $s, s+1, \ldots, n, 1, 2 \ldots, s-1$, where, s is a shift parameter much larger than k.

consistently superior to the absolute deltoids algorithm. The results are presented in Figure 2.

Experiment 2. In this experiment, we evaluate the COUNTSKETCH DYADIC, COUNTSKETCH LINEAR and the variational deltoids algorithm. We consider data whose frequency is distributed as zipfian difference *zipfdiff*(z, z), for parameters $z = 0.3, 0.4$ and 0.5. The number of distinct items was fixed at 4 million items. The total space used by the algorithms is given in Figure 3 and varies between 2.5— 10% of the space required to actually store the data. In comparison, in experiment 1, it was varied between 10 — 40% of the size of the data. Thus, the experiments in this category use significantly less space (percentage wise) than the first experiment and significantly stresses the retrieval capabilities of the algorithms. The parameter choices are as follows. For COUNTSKETCH DYADIC, the settings are the same as those of COUNT-MIN DYADIC wherever possible. That is, the number of random permutations used is 1, the number of levels is kept between 19 and 21 and the number of tables is kept between 5 and 7. Recall that for the COUNTSKETCH LINEAR algorithm, s_2 is the number of sketches in each group whose average (of the squares) is taken, and s_3 is the number of such groups; for each bit value 0 or 1, for each bit position 1 through $\log n$ and each bucket of each table. In our experimentation, s_2 is set to 1 and s_3 to 5. These settings are significantly smaller than the theoretical bounds. For the variational deltoids algorithm, the number of tables were kept between 5 and 7. Since the space provided to the algorithms is the same, the main parameter that varies is the height of each of the tables, subject to the above settings.

Results of Experiment 2. The results of the experiments are summarized in Figure 3. Corresponding to each of the three algorithms tested, the precision and recall are shown in the same column (except when recall is 0). The nature of the results are both surprising and conclusive. It appears that COUNTSKETCH DYADIC is significantly superior in terms of both precision and recall to the COUNTSKETCH LINEAR algorithm, whereas the performance of the variational deltoids algorithm is quite poor. The recall is not 100%, given that the space provided to the algorithms is very small. Further, as expected, both precision and recall improve with increased space. It is an unexpected observation that COUNTSKETCH DYADIC is substantially superior to the other two algorithms.

7 Conclusions

We present novel and practical space and time-efficient algorithms for finding frequent items, absolute range sums and absolute quantiles over general streams.

Acknowledgements

We thank Tejas Gandhi and M. Ravibabu for implementing the reversible hashing algorithm of [11].

Distribution	Space (in size of) (doubles)	Threshold αF_1 α	Actual No of frequent items	Recall Absolute Deltoids [4]	Recall Count-Min Dyadic
zipfdiff $(0.1, 0.9)$	210540	2^{-9}	11	9	10
		2^{-10}	20	14	16
		2^{-11}	40	19	24
	409600	2^{-9}	11	10	11
		2^{-10}	20	17	17
		2^{-11}	40	24	29
		2^{-12}	86	37	52
	778240	2^{-9}	11	11	11
		2^{-10}	20	18	20
		2^{-11}	40	29	32
		2^{-12}	86	49	61
		2^{-13}	179	73	100
zipfdiff $(0.4, 0.5)$	210540	2^{-9}	0	0	0
		2^{-10}	0	0	0
		2^{-11}	0	0	0
	409600	2^{-9}	0	0	0
		2^{-10}	0	0	0
		2^{-11}	0	0	0
		2^{-12}	3	1	1
	778240	2^{-9}	0	0	0
		2^{-10}	0	0	0
		2^{-11}	0	0	0
		2^{-12}	3	1	2
		2^{-13}	8	6	11
zipfdiff $(0.3, 0.7)$	210540	2^{-9}	3	2	3
		2^{-10}	7	4	4
		2^{-11}	13	5	8
	409600	2^{-9}	3	3	3
		2^{-10}	7	4	4
		2^{-11}	13	8	9
		2^{-12}	26	11	16
	778240	2^{-9}	3	3	3
		2^{-10}	7	5	4
		2^{-11}	13	10	11
		2^{-12}	26	16	18
		2^{-13}	72	22	26

Fig. 2. F_1-based frequent items: Comparing absolute deltoids method [4] with Count-Min Dyadicmethod. Number of items = 2^{21}.

Distribution	Space (in size of) (doubles)	Threshold $(\alpha F_2)^{1/2}$ α	Actual No of frequent items	Recall, Precision Variational Deltoids [4]	Recall, Precision CountSketch Dyadic	Recall, Precision CountSketch Linear
zipf diff(0.3, 0.3)	307240	2^{-9}	2	0	0, 0	1,0
		2^{-10}	8	0	3, 3	2,1
		2^{-11}	24	0	4, 4	3,1
		2^{-12}	76	0	10, 8	3,1
		2^{-13}	232	0	26, 19	3,1
	573440	2^{-9}	2	0	0, 0	0
		2^{-10}	8	0	4, 4	0
		2^{-11}	24	0	7, 7	0
		2^{-12}	76	0	18, 18	1,0
		2^{-13}	232	0	38, 37	1,0
	1064960	2^{-9}	2	0	0, 0	1,1
		2^{-10}	8	0	4, 4	1,1
		2^{-11}	24	0	10, 10	3,2
		2^{-12}	76	0	26, 26	3,2
		2^{-13}	232	0	54, 53	3,2
zipf diff(0.4, 0.4)	307240	2^{-9}	17	0	8, 8	5,5
		2^{-10}	42	0	19, 19	7,7
		2^{-11}	99	0	39, 39	8,8
		2^{-12}	232	0	60, 59	10,9
		2^{-13}	540	0	115, 96	10,9
	573440	2^{-9}	17	2,2	11, 11	6, 6
		2^{-10}	42	3,3	24, 24	6, 6
		2^{-11}	99	0	44, 44	6, 6
		2^{-12}	232	0	91, 91	7,7
		2^{-13}	540	0	154, 149	7,7
	1064960	2^{-9}	17	6	12, 12	16, 14
		2^{-10}	42	8	28, 28	21, 19
		2^{-11}	99	2	56, 56	21, 20
		2^{-12}	232	0	109, 109	22, 22
		2^{-13}	540	0	184, 184	24, 24
zipf diff(0.5, 0.5)	307240	2^{-9}	42	10, 10	27, 27	8, 7
		2^{-10}	84	4, 4	50, 50	9, 8
		2^{-11}	167	0	77, 77	9, 9
		2^{-12}	334	0	125, 122	9, 9
		2^{-13}	644	0	210, 183	10, 10
	573440	2^{-9}	42	14, 14	29, 29	25, 22
		2^{-10}	84	16, 16	56, 56	29, 28
		2^{-11}	167	3 , 3	95, 95	30, 30
		2^{-12}	334	0	162, 162	31,31
		2^{-13}	644	0	256, 256	31, 31
	1064960	2^{-9}	42	16,16	37, 37	31, 28
		2^{-10}	84	26,26	66, 66	41, 39
		2^{-11}	167	20,20	119, 119	44, 42
		2^{-12}	334	7, 7	208, 208	47, 44
		2^{-13}	644	1, 1	359, 359	48, 46

Fig. 3. F_2-based frequent items: Comparing CountSketch Dyadic, CountSketch Linear and variational deltoids

References

1. Noga Alon, Yossi Matias, and Mario Szegedy. "The space complexity of approximating frequency moments". *J. Comp. Sys. and Sc.*, 58(1):137–147, 1998.
2. Moses Charikar, Kevin Chen, and Martin Farach-Colton. "Finding frequent items in data streams". In *Proc. ICALP, 2002*, pages 693–703.
3. Graham Cormode and S. Muthukrishnan. "An Improved Data Stream Summary: The Count-Min Sketch and its Applications". *J. Algorithms*, 55(1).
4. Graham Cormode and S. Muthukrishnan. "What's New: Finding Significant Differences in Network Data Streams". In *Proc. IEEE INFOCOM*, 2004.
5. E. D. Demaine, A. López-Ortiz, and J. I Munro. "Frequency estimation of internet packet streams with limited space". In *Proc. ESA*, pages 348–360, 2002.
6. Anna Gilbert, Y. Kotidis, S. Muthukrishnan, and Martin Strauss. "How to Summarize the Universe: Dynamic Maintenance of Quantiles". In *Proc. VLDB*, pages 454–465, Hong Kong, August 2002.
7. E. Kaplan, M. Naor, and O. Reingold. "Derandomized Constructions of k-Wise (Almost) Independent Permutations". In *Proc. RANDOM, 2005*, pages 354–365.
8. R.M. Karp, S. Shenker, and C.H. Papadimitriou. "A Simple Algorithm for Finding Frequent Elements in Streams and Bags". *ACM TODS*, 28(1):51–55, 2003.
9. M. Luby and C. Rackoff. "How to construct pseudorandom permutations and pseudorandom functions". *SIAM J. Comp.*, 17(1):373–386, 1988.
10. J. Misra and Gries. D. "Finding repeated elements". *Sci. Comput. Programm.*, 2:143–152, 1982.
11. R. Schweller, Z. Li, Y. Chen, Y. Gao, A. Gupta, Y. Zhang, P. Dinda, M-Y. Kao, and G. Memik. "Monitoring Flow-level High-speed Data Streams with Reversible Sketches". In *Proc. IEEE INFOCOM*, 2006.
12. M. Thorup and Y. Zhang. "Tabulation based 4-universal hashing with applications to second moment estimation". In *Proc. ACM SODA*, pages 615–624, New Orleans, Louisiana, USA, January 2004.