Min-wise independent sampling from skewed data streams

Konstantin Kutzkov
IT University of Copenhagen, Denmark
konk@itu.dk

Abstract

Min-wise independent hashing is a powerful sampling technique for estimating the similarity between sets. In particular, it has proved to be ubiquitous for mining data streams of large volume where the input sets are revealed in arbitrary order and the elements in a given set do not arrive consecutively. More precisely, for sets of elements $E$ and attributes $A$ the input is a stream of element-attribute pairs $(e,a)$, $e \in E$, $a \in A$, arriving in arbitrary order. One is interested in detecting elements with similar sets of attributes.

We present a new technique for applying min-wise independent hashing to streamed data sets when the frequency distribution of the number of elements in the sets is highly skewed, a common situation for many real-life applications, and one in obtaining samples of attributes only for the most frequent elements. The approach combines classic min-wise independent hashing with sketch-based frequent items mining algorithms. We prove that it is more space efficient than na"ıve subsampling when the numbers of attributes per element adhere to Zipfian distribution with parameter $z > 1$ and discuss applications in graph stream mining and association rule mining in transactional streams.

1 Introduction

Min-wise independent hashing was first proposed [3] as an efficient solution for the detection of nearly duplicate documents. In a nutshell, it works by choosing a random permutation $\pi$ of the bag of words, then computing a signature for each document based on $\pi$ and comparing the signatures. However, choosing uniformly at random from the set of all permutations requires to explicitly store the rank of each element given by $\pi$, i.e., an exponential number of bits. Instead, Broder et al. [3] experimentally showed that a permutation given by the relative order defined by a pairwise independent hash function yields good results. Indyk [13] was the first to show how to obtain an approximately random permutation using efficiently computable hash functions. These results made min-wise independent hashing the ubiquitous choice when estimating the similarity between sets revealed in a streaming fashion.

There are two variations of the approach corresponding to sampling with and without replacement. Assume we want to estimate the Jaccard similarity $^1$ between two sets $A$ and $B$ and we need $k$ samples from each of them. In the first variant we assume there are $k$ permutations $\pi_i$ and select the first element in $A$ and $B$ for each $\pi_i$. The expectation of the indicator random variable comparing the two elements is exactly $J(A, B)$, thus the average of $k$ such estimators yields a precise estimate for a sufficiently big $k$.

Another variation works with a single permutation $\pi$ and stores the $k$ smallest element in $A$ and $B$ with respect to $\pi$, denote them by $\pi_k(A)$ and $\pi_k(B)$, respectively, from which one then computes $\pi_k(A \cup B)$ the $k$ smallest elements in $A \cup B$. The similarity is then estimated as $\frac{|\pi_k(A \cup B) \cap \pi_k(A) \cap \pi_k(B)|}{k}$. The disadvantage of this variant is that the permutation needs to be “more random” than when taking the average estimate yielded by $k$ different permutations chosen uniformly at random from a suitably defined family but on the other hand, when applied in a streaming setting, it allows faster processing time per element [10].

$^1$For two sets $A$ and $B$, the Jaccard similarity is defined as $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$. 

1
Uneven distribution of the elements in real data is ubiquitous, be it the in-degree distribution of the Web graph [15] or the number of bought items in market baskets [14], therefore researchers have addressed the problem by designing algorithms working under the assumption that elements in the input data adhere to some form of power law. For example, Cormode and Muthukrishnan [9] showed that assuming Zipfian distribution for the item frequencies in data streams, the COUNT-MIN [8] algorithm can detect the $k$ most frequent items using asymptotically less space than when simply assuming a lower bound on the frequency of the $k$ most frequent items. In a later work Berinde et al. [2] showed that even better guarantees can be obtained for deterministic frequent item mining algorithms [16, 17].

Motivated by practical applications, in the present paper we propose a new approach for obtaining the min-wise independent samples of the largest sets revealed in a streaming fashion when the set sizes differ significantly.

**Problem statement and applications.** The input is a stream $S$ of element-attribute pairs $(e, a)$. Let $A_e = \{a : (e, a) \in S\}$, i.e., the set of attributes of the element $e$. We are interested in detecting pairs of elements that occur in sufficiently many element-attribute pairs and are similar to each other, i.e., the Jaccard similarity of their sets of attributes is above a certain threshold.

The problem has applications in graph mining and association rule mining in data streams. Consider first the problem of detecting vertices with similar neighborhoods in streamed graphs. Let $S$ be a simple directed graph $G$ provided as a stream of edges $(u, v)$ in arbitrary order denoting that there is an edge from $u$ to $v$. The in-neighborhood of a vertex $u$ is the set $N_u^n = \{v \in V : (u, v) \in G\}$, and similarly the out-neighborhood of $u$ is $N_u^\text{out} = \{v \in V : (u, v) \in G\}$. With each directed edge we associate an element-attribute pair. Depending on the application, we can mine for vertices with similar in- or out-neighborhoods, thus $u$ can be the element and $v$ the attribute, or vice versa. Min-wise independent hashing has been used for graph clustering and compression [4, 19] as well as for the detection of dense subgraphs [11]. At a high level, all these algorithms aim at grouping together vertices with similar neighborhoods. Therefore, it is a natural and practically motivated extension to compress and cluster together only high degree vertices since these vertices are incident to the majority of the edges.

As another example consider the fundamental data mining problem of detecting interesting pairs of items in transactional data streams. In this problem one is given a ground set $I$ of $n$ items and a stream $S$ of $m$ transactions $T_i \subseteq I$. Here we identify elements with items and the attributes are transaction IDs. For example $I$ is the set of goods offered in a supermarket and transactions are customers shopping baskets. One is then interested in detecting item pairs whose occurrences in transactions appear to be correlated. Such pairs are then used to define association rules among items. For example, “customers who buy beer are likely to also buy chips”. Cohen et al. [7] applied min-wise independent hashing to detect similar pairs of (not necessarily frequent) elements by storing a min-wise sample for each element in the stream. However, for many applications one also wants such association rules to be representative and therefore one considers only frequent pairs that occur in a sufficiently big number of transactions. Classic approaches like Apriori [1] and FP-growth [12] work by first determining the set of frequent pairs and then deducing association rules from them but one needs two passes over the transactions, and thus cannot handle transactional streams in real time. Toivonen [20] proposed an intuitive algorithm that samples individual transactions. One can show that for a sufficiently big number of sampled transactions, all similar pairs of frequent items can be detected with high probability, the best known bounds are in [5].

2 Notation

Our algorithm works with hash functions which will provide a permutation of the elements in a given set which is “close enough” to a permutation chosen at random from the set of all possible permutations. The necessary definitions are provided below.

A family $\mathcal{F}$ of functions from $U$ to $Z$ is $k$-wise independent if for $f : U \to Z$ chosen uniformly at random from $\mathcal{F}$

$$\Pr[f(u_1) = c_1 \land f(u_2) = c_2 \land \cdots \land f(u_k) = c_k] = z^{-k},$$
for $z = |Z|$, distinct $u_i \in U$ and any $c_i \in Z$ and $k \in \mathbb{N}$. (We will also say that $f$ is $k$-wise independent."

A family $\mathcal{H}$ of functions from $U$ to a finite totally ordered set $S$ is called \textit{(\(\alpha, k\))-min-wise independent} \cite{10} if for any $X \subseteq U$ and $Y \subseteq X$, with $|Y| = k$ and $0 < \alpha < 1$, for a function $h$ chosen uniformly at random from $\mathcal{H}$ it holds
\[
\Pr[\max_{y \in Y} \{h(y)\} < \min_{z \in X \setminus Y} \{h(z)\}] = (1 + \alpha) \frac{1}{\binom{|X|}{k}}.
\]

We will also use \textit{tabulation hashing} in our algorithm. For a function $h : V \rightarrow [s]$ we view each key $v \in V$ as a vector consisting of $c$ characters, $v = (v_1, v_2, \ldots, v_c)$, where the $i$th character is from a universe $V_i$ of size $n^{1/c}$. (Without loss of generality we assume that $n^{1/c}$ is an integer.) For each universe $V_i$ we initialize a table $T_i$ and for each character $v_i \in V_i$ we store a random value $r_{v_i} \in [s]$. Then the hash value is computed as
\[
h_0(v) = T_1[v_1] \oplus T_2[v_2] \oplus \ldots \oplus T_c[v_c]
\]
where $\oplus$ denotes the bit-wise xor operation. Thus, for a (small) constant $c$, the space needed is $O(n^{1/c} \log n)$ bits and the evaluation time is $O(1)$ array accesses. In the following we assume $c \geq 2$.

We use the notation $h : U \rightarrow [0, 1]$ to denote that $h$ maps $U$ to a finite subset $D$ of $[0, 1]$. For $h$ being pairwise independent and $|D| = |U|^3$, the probability of collision, i.e., $h(u_1) = h(u_2)$ for $u_1, u_2 \in U$, $u_1 \neq u_2$, is at most $1/|U|$. We thus assume that $h : U \rightarrow [0, 1]$ is injective with high probability and $h(u)$ can be described using $O(\log |U|)$ bits for $u \in U$. In the following we will check for equality attribute hash values instead of attributes.

We will say that an algorithm returns an \((\epsilon, \delta)\)-approximation of some quantity $q$ if it returns a value $\tilde{q}$ such that $(1 - \epsilon)q \leq \tilde{q} \leq (1 + \epsilon)q$ with probability at least $1 - \delta$ for every $0 < \epsilon, \delta < 1$.

\textbf{Skewed distribution.} We define frequency of an element $a$ as the number of element-attribute pairs $(e, a)$ occurring in the stream. A common formalization of the skew in data is the assumption of Zipfian distribution. Assuming a total order on the elements given by decreasing frequency, the elements in a given set $M$ over $N$ elements follow Zipfian distribution with parameter $z > 0$ if the frequency of the element of rank $i$ is $\frac{|M|}{\zeta(z)^i}$ where $\zeta(z) = \sum_{i=1}^{N} i^{-z}$ and $|M|$ denotes the total weight of elements in $M$. We will use the fact that for $z > 1$, $\sum_{i=b+1}^{N} i^{-z} = O(b^{1-z})$.

\section{The algorithm}

If we knew which are the frequent elements before processing the stream, then the problem becomes trivial. For a persistently stored database of element-attribute pairs a possible solution would be to determine the frequent elements in a first pass, and to compute the samples in a second pass. For many real-life applications it is essential to process the input stream in a single pass.

A straightforward approach for computing a min-wise sample of the attributes of a frequent element is to fix a threshold $\alpha \in (0, 1]$ and keep each element-attribute pair $(e, a)$ such that $h(a) \leq \alpha$ for a uniform hash function $h : A \rightarrow [0, 1]$. Clearly, for an element which occurs $k$ times, we expect $\alpha k$ sampled attributes. However, for highly skewed data the obvious disadvantage of the approach is that we will sample more than necessary attributes for the most frequent elements and this will dominate the space cost.

At a high level, our approach extends hashing based frequent items mining algorithms \cite{6, 8} to detecting the min-wise independent samples of the most frequent elements. These algorithms distribute the items to $b$ different buckets by a suitably defined hash function and maintain a counter for each bucket. Depending on how the counter is updated, one can obtain an additive approximation of the frequency of each item in terms of 1- or 2-norm of the frequency vector of the stream and the number of buckets $b$. Instead of maintaining a counter in each bucket, we will store a set of elements and their attributes with the smallest hash values. We show that for the heaviest elements we will obtain the required number of attribute hash values with good probability.

In Figure 1 we present the new algorithm for computing min-wise independent samples for skewed data. We assume that the input is a stream of pairs $(e, a)$ where $e \in E$ for a set of elements $E$ and $a \in A$ for a
set of attributes $A$. The core of the algorithm is the procedure $\text{COLLECTMINVALUES}$. We assume access to a hash function $g : E \rightarrow [b]$ implemented by tabulation hashing with constant $c$ and a $t$-wise independent $h : A \rightarrow [0, 1]$. Each incoming pair $(e, a)$ is distributed to a bucket $B_e$ by $g(e)$. In a bucket we keep track of the $\kappa$ attributes $a_i$ with smallest hash values $h(a_i)$, denote it as $A_\kappa$, for a user-defined $\kappa$. (We assume that $h$ is injective with high probability.) We record all element-hash value pairs $(e, h(a_i))$ such that $a_i \in A_\kappa$. At the end, for each element $e$ with at least $s$ records $(e, h(a_i))$ in $B_e$ we return the set of hash values $MWS_e = \{h(a_i) : a_i \in A_\kappa\}$.

In order to guarantee that for each heavy element we will detect the required $s$ smallest hash values, we will run the above in parallel for different hash functions $g$ and $h$. This is done in $\text{HEAVYELEMENTSKETCHES}$ where we assume as input two sets of hash functions $\mathcal{G}$ and $\mathcal{H}$. Finally, $\text{SIMILARITYESTIMATOR}(e_1, e_2)$ returns an estimation of the similarity between two elements $e_1$ and $e_2$. This is done by computing the estimate from the number of common attributes among the $k$ attributes with the $k$ smallest hash values in $A_{e_1} \cup A_{e_2}$, if possible. If there are at least $t$ estimates from the functions in $\mathcal{H}$, we return the median of them.

In the next section we obtain bounds on the parameters $s, \kappa$ and $t$ and the required number of functions in $\mathcal{G}$ and $\mathcal{H}$, such that the algorithm returns an $(\varepsilon, \delta)$-approximation of the Jaccard similarity of all pairs among the heaviest $k$ elements.

**COLLECTMINVALUES**

**Input:** stream $S$ of element-attribute pairs $(e, a)$, $e \in E, a \in A$, hash functions $g : E \rightarrow [b]$, $h : A \rightarrow [0, 1]$, capacity $\kappa$, threshold $s$

1: initialize an array of $b$ buckets
2: for each $(e, a)$ in $S$ do
3: \quad $B = g(e)$
4: \quad if $h(a)$ is smaller than the largest hash value of an attribute in bucket $B$ then
5: \quad \quad $B = B \cup (e, h(a))$
6: \quad if there are more than $\kappa$ attributes in $B$ then
7: \quad \quad update $B$ to contain the $\kappa$ attributes with the smallest hash values.
8: for $B \in [b]$ do
9: \quad if there are more than $s$ pairs $(e, h(a))$ for some element $e$ then
10: \quad \quad return the $s$ smallest values $h(a)$ of $e$
11: \quad else
12: \quad \quad return $\emptyset$

**HEAVYELEMENTSKETCHES**

**Input:** stream $S$ of element-attribute pairs $(e, a)$, $e \in E, a \in A$, family $\mathcal{G}$ of hash functions $g : E \rightarrow [b]$, family $\mathcal{H}$ of hash functions $h : A \rightarrow [0, 1]$, capacity $\kappa$, threshold $s$, threshold $k$

1: for $g \in \mathcal{G}$ do
2: \quad for $h \in \mathcal{H}$ do
3: \quad \quad $\{MWS_{g,h}^i\}_{i \in E} = \text{COLLECTMINVALUES}(S, g, h, \kappa, s)$

**SIMILARITYESTIMATOR**

**Input:** element $e_1$, element $e_2$, thresholds $s$ and $t$

1: for $g \in \mathcal{G}$ do
2: \quad for $h \in \mathcal{H}$ do
3: \quad \quad if $MWS_{g,h}^{e_1} \neq \emptyset$ and $MWS_{g,h}^{e_2} \neq \emptyset$ then
4: \quad \quad \quad $MWS_{g,h}^{e_1, e_2}(s) = (MWS_{g,h}^{e_1} \cup MWS_{g,h}^{e_2})$
5: \quad \quad \quad Estimates = Estimates $\cup \{MWS_{g,h}^{e_1, e_2}(s) \cap MWS_{g,h}^{e_1} \cap MWS_{g,h}^{e_2}\}/s$
6: if $|\text{Estimates}| \geq t$ then
7: \quad \quad return median($\text{Estimates}$)

Figure 1: Pseudocode description of the algorithm.
4 Analysis

We will analyze the algorithm for Zipfian distribution with parameter $z > 1$. The next lemma shows that for a sufficiently random hash function $h : A \rightarrow [0,1]$ with good probability one can obtain an $(1 \pm \varepsilon)$-approximation of the Jaccard similarity between two attribute sets using a number of samples that is close to the one required for a truly random permutation. In the following we use $J(e_1, e_2)$ instead of $J(A_{e_1}, A_{e_2})$.

**Lemma 1** Let $e_1$ and $e_2$ be two elements such that $J(e_1, e_2) \geq \tau$. Let $h : A \rightarrow [0,1]$ be an $(1/9, s)$-independent hash function and $A_{e_1}^h$ and $A_{e_2}^h$ by the $s$ elements with the $s$ smallest hash values in $A_{e_1}$ and $A_{e_2}$, respectively. Then for $s = O(\frac{1}{J^2})$, $\hat{J} = \frac{|(A_{e_1} \cup A_{e_2}) \cap A_{e_1}^h \cap A_{e_2}^h|}{|A_{e_1} \cup A_{e_2}|}$ is an $(1 \pm \varepsilon)$-approximation of $J(e_1, e_2)$ with probability at least $2/3$. If $J(e_1, e_2) < \tau$, then $\hat{J} < (1 + \varepsilon)\tau$ with probability at least $7/12$.

**Proof:** Assume we have a truly random permutation of $A$ and let $(A_{e_1} \cup A_{e_2})^*$ be the first $s$ attributes with respect to this permutation. Let $t = |A_{e_1} \cup A_{e_2}|$. Let $X$ be a random variable counting the number of attributes from $A_{e_1} \cap A_{e_2}$ in $(A_{e_1} \cup A_{e_2})^*$. Clearly, it holds $E[X] = Js$. The number of attributes from $A_{e_1} \cup A_{e_2}$ in size-$s$ subsets follow hypergeometric distribution with $s$ samples from a set of size $t$ and $Jt$ successes. Thus,

$$V[X] = J(1 - J)s \frac{t - s}{t - 1} < Js$$

for $s > 1$. By Chebyshev’s inequality we thus have

$$\Pr[X \neq (1 \pm \varepsilon)E[X]] \leq \frac{V[X]}{\varepsilon^2 Js^2} < \frac{1}{\varepsilon^2 Js}.$$

Consider first the case that $J \geq \tau$. For $s = \frac{4}{J^2}$ the probability that the value $X/s$ is not an $(1 \pm \varepsilon)$-approximation of $J$ is at most $1/4$.

Since $h$ is $(1/9, s)$-independent, the probability that the $s$ smallest hash values correspond to a “good” size-$s$ subset, i.e., a subset yielding a $(1 \pm \varepsilon)$-approximation, is at least $2/3$, and to “bad” size-$s$ subset – at most $5/18$. This yields the claimed probability that the returned value is not an $(1 \pm \varepsilon)$-approximation. Similarly, if $J < \tau$ we can bound the probability that we return a value larger than $(1 + \varepsilon)J$ to $5/12$. □

The following theorem is our main result.

**Theorem 1** Let $E$ be a set of $n$ elements and $A$ be a set of $m = O(n)$ attributes. Let $S$ be a stream of element-attribute pairs $(e, a), e \in E, a \in A$. Assume that the element frequencies in $S$ are distributed according to Zipfian distribution with parameter $z > 1$ and let $Top_k$ be the $k$ most frequent elements in $S$ for $n^\beta \leq k \leq n$, for any $\beta > 0$. Let $\ell \leq m$ be the maximum number of elements-attribute pairs an attribute can occur in. For $e_1, e_2 \in Top_k$ and user defined $\varepsilon, \delta, \tau \in (0,1)$, there exists a one-pass streaming algorithm which returns a value $\hat{J}$ such that

- If $J(e_1, e_2) \geq \tau$, then $\hat{J}$ is an $(\varepsilon, \delta)$-approximation of $J(e_1, e_2)$.
- Otherwise, if $J(e_1, e_2) < \tau$, then $\hat{J} \leq (1 + \varepsilon)\tau$ with probability at least $1 - \delta$.

The algorithm processes each element-attribute pair in time $O(\log^2(\frac{k}{\varepsilon}) \log \frac{1}{\tau \varepsilon})$ and uses $O((k + \ell) \frac{1}{\varepsilon} \log^2(\frac{k}{\varepsilon}) \log n)$ bits of space.

**Proof:** We will show that the algorithm presented in the previous section returns a value which satisfies the claims in the theorem. The analysis consists of two parts. We first show that implementing the distribution function $g$ by tabulation hashing, elements will be evenly distributed among buckets with high probability and obtain bounds on the required number of hash functions in $G$. Then, under the assumption that elements are well distributed among buckets, we will show that with good probability we will obtain an $(1 \pm \varepsilon)$-approximation of all pairs of heavy elements with Jaccard similarity at least $\tau$. For a sufficiently big number of functions in $H$, the success probability will be then amplified to any $1 - \delta$, for a user-defined $\delta$. 

5
Consider a given hash function $g : E \to [b]$ implemented by tabulation hashing with parameter $c$ for constant $c > 2$. Assume for the number of buckets it holds $b = 8k$. Consider a given element $e \in \text{Top}_k$ and let $B_e = g(e)$. Since $g$ is $3$-wise independent [18], each $e' \in \text{Top}_k \setminus \{e\}$ is hashed to $B_e$ with probability $\frac{1}{8k}$ and by Markov’s inequality we bound $\Pr[3e' \in \text{Top}_k \setminus \{e\} : g(e') = B_e] \leq 1/8$, i.e., with probability at most $1/8$ there is another heavy element in $B_e$. Using the fact that $\sum_{i=k+1}^{|E|} i^{-z} \leq 2k^{1-z}$ for $z > 1$, we expect non-frequent elements with a total number of attributes $\frac{|E|}{\kappa(z)k}$ to be hashed by $g$ in $B_e$. Thus, again by Markov’s inequality, the total number of attributes from non-heavy elements is bounded by $\frac{|E|}{\kappa(z)k^c}$ with probability $1/4$. Therefore, we conclude that with probability at least $5/8$ the total number of attributes of elements different from $e$ is bounded by $\frac{|E|}{\kappa(z)k^c}$.

For $O(\log k/\delta)$ functions in $G$ chosen uniformly at random from a $3$-wise independent family, the probability that the total weight of attributes in $B^z_e$ is more than $\frac{2|E|}{\kappa(z)k}$ for all functions $g$ can thus be bounded to $\delta/(2k)$. Thus, by the union bound, the statement holds for all elements $e \in \text{Top}_k$ with probability $1 - \delta/2$.

Denote by $E_a$ the set of elements that have $a$ as an attribute. We assume $|E_a| \leq \ell$ for all $a \in A$. We will use the result by Pătraşcu and Thorup on the holds of tabulation hashing [18]. For $\ell > b$, no bucket gets more than

$$\ell/b \pm O(\sqrt{\ell/b} + \log^c \ell)$$

elements from $E_a$ with probability $1 - \ell^{-\gamma}$ for any constant $\gamma > 0$. For $\ell \leq b$ no buckets gets more than a small constant number of elements from $E_a$ with probability $1 - b^{-\gamma}$ for any constant $\gamma > 0$. Since we assume $b = 8k = O(n^\beta)$ for some $\beta > 0$, it follows that with high probability no bucket will contain more than $O(\ell/k)$ elements from $E_a$, for all $a \in A$.

In the following we assume that the total number of attributes in $B_e$ is at most $\frac{2|E|}{\kappa(z)k}$, and at least $\frac{|E|}{\kappa(z)k}$ of these are for attributes for the element $e$. Let $s$ be the number of required samples (to be determined later) and let the capacity $\kappa$ of each bucket be $3s$. Thus, we expect $3s/2$ of the smallest hash values in the bucket to be for $e$’s attributes. Using the same reasoning as in the proof of Lemma 1, we thus obtain that for $h$ being $(1/8, 3s)$-independent, we can bound the probability that among the $3s$ smallest hash values in $B_e$ there are less than $s$ hash values of attributes of $e$ to be less than $1/16$. Thus, for a given pair of heavy elements $u$ and $v$, with probability at least $7/8$ we have found the $s$ smallest hash values for $u$ and $v$. By Lemma 1, for $s = O(1/\tau)$ and $J(u,v) \geq \tau$ with probability at least $2/3$, $|\text{MWS}^h_{e_1, e_2}(s) \cap \text{MWS}^h_{e_1} \cap \text{MWS}^h_{e_2}|/s$ will be an $(1 \pm \epsilon)$-approximation of $J(e_1,e_2)$, and for $J(e_1,e_2) < \tau$ it holds

$$\Pr[|\text{MWS}^h_{e_1, e_2}(s) \cap \text{MWS}^h_{e_1} \cap \text{MWS}^h_{e_2}|/s \geq (1 + \epsilon)J(u,v)] \geq 2/3.$$  

Thus, by the union bound, with probability more than $1/2$ the claim in the theorem holds. A standard application of Chernoff’s inequality yields that the median of $t = O(\log \frac{4}{\delta})$ such estimators is an $(1 \pm \epsilon)$-approximation with probability at least $1 - \delta/k^2$, thus we need $|\mathcal{H}| = O(\log \frac{4}{\delta})$. Summing up, for each pair of heavy elements $e_1$ and $e_2$ we find the required $s = O(\frac{1}{\tau^2})$ minimum hash values in $A_{e_1} \cup A_{e_2}$ with probability $1 - \delta/2$ and they yield a value as desired with probability $1 - \delta/2$, thus by the union bound the claimed approximation guarantee holds.

We next analyze the complexity of the algorithm. The hash functions in $G$ can be each described in $O(n^{1/c})$ machine words, for any constant $c > 1$, and evaluated in $O(1)$ time. For each attribute $a$ recorded in a bucket, there are $O(\ell/k + 1)$ pairs $(e, h(a))$, thus in total for all buckets $O(sk(\ell/k + 1)) = O(s(\ell + k))$. For the $(O(1), O(s))$-independent hash functions in $\mathcal{H}$ we use the construction of Feigenblat et al. [10]. This yields that each $h \in \mathcal{H}$ can be described in $O(s)$ machine words and evaluated in time $O(\log s)$. Implementing each bucket as a binary search tree would guarantee an update time of $O(\log s)$ for each new incoming pair.

The space for the hash functions $G$ and $\mathcal{H}$ is $O(\log \frac{4}{\delta} \log n(n^{1/c} + s))$ bits for arbitrary constant $c > 2$. Each copy of \textsc{CollectMinValues} needs $O(s(\ell + k) \log n)$ bits, and there are $O(\log^2 \frac{4}{\delta})$ such copies. Since $k \geq n^\beta$ for some constant $\beta > 0$, the space amounts to $O(s(\ell + k) \log^2 \frac{4}{\delta} \log n)$. The processing time per element-attribute pair is $O(\log^2 \frac{4}{\delta} \log s)$. 

\[\square\]
On sampling from transactional streams. Note that the above analysis assumes that element-attribute pairs arrive in arbitrary order. However, in transactional streams all elements associated with a given attribute, i.e., the transaction ID, arrive consecutively. Thus, one can simply generate a random number in \([0, 1]\) instead of using a hash function \(h : A \rightarrow [0, 1]\).

Comparison to subsampling. The space complexity of the naïve subsampling algorithm in order to obtain the required number of sampled element-attribute pairs for the \(k\) heaviest elements with high probability, is \(O(k^2 \frac{1}{\epsilon^2} \log n)\) for Zipfian distribution with \(z > 1\). In particular, this is implied by the best known bounds on transaction sampling \([5]\). Note that in this case \(\ell\) is the maximum transaction width and for many domains this is a large number, see e.g. \([14, 15]\).

References

