Exponential Time Improvement for *min-wise* Based Algorithms*

Guy Feigenblat† Ely Porat ‡ Ariel Shiftan §¶

Abstract

In this paper we extend the notion of *min-wise* independent family of hash functions by defining a *k-min-wise* independent family of hash functions. Informally, under this definition, all subsets of size k of any fixed set X have an equal chance to have the minimal hash values among all the elements in X, when the probability is over the random choice of hash function from the family. This property measures the randomness of the family, as choosing a truly random function, obviously, satisfies the definition for $k = |X|$. We define and give an efficient time and space construction of approximately *k-min-wise* independent family of hash functions by extending Indyk's construction of approximately *min-wise* independent [1]. The number of words needed to represent each function is $O(k \log\log(\frac{1}{\epsilon}) + \log(\frac{1}{\epsilon}))$, which is only suboptimal by a factor of $O(\log\log(\frac{1}{\epsilon}))$, where $\epsilon \in (0, 1)$ is the desired error bound. This construction is the first applicable for sampling bottom-k sketches [2, 3] out of the universe. In addition, we introduce a general and novel technique that utilizes our construction, and can be used to improve many *min-wise* based algorithms, such as [4, 5, 6, 7, 3, 2, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]. As an example we show how to apply it for similarity estimation over data streams, and reduce exponentially the run time of the current known result [5]. In addition, we also discuss improvements of known algorithms for estimating rarity and entropy of random walk over graphs (from SODA07 [20]).

1 Introduction

Hash functions are fundamental building blocks of many algorithms. They map values from one domain to another, usually smaller. Although they have been studied for many years, designing hash functions is still a hot topic in modern research. In a perfect world we could use a truly random hash function, one that would be chosen randomly out of all the possible mappings.

Specifically, consider the domain of all hash functions $h : N \to M$, where $|N| = n$ and $|M| = m$. As we need to map each of the n elements in the source into one of the m possible mappings, the number of bits needed to maintain each function is $n \log m$. Since nowadays we often have massive amount of data to process, this amount of space is not feasible. Nevertheless, most algorithms do not really need such a high level of randomness, and can perform well enough with some relaxations. In such cases one can use a much smaller domain of hash functions. A smaller domain implies lesser space requirement at the price of a lower level of randomness.

As an illustrative example, the notion of *2-wise-independent* family of hash functions assures the independence of each pair of elements. It is known that only $2 \log m$ bits are enough in order to choose and maintain such a function out of the family.

This work is focused on the area of min-hashing. One derivative of min-hashing is *min-wise* independent permutations, which were first introduced in [21, 22]. A family of **permutations** $F \in S_n$ (where S_n the symmetric group) is **min-wise independent** if for any set $X \subseteq [n]$ (where $[n] = \{0, \ldots, n-1\}$) and any $x \in X$, where π is chosen uniformly at random in F, we have:

$$Pr[\min\{\pi(X)\} = \pi(x)] = \frac{1}{|X|}$$

Similarly, a family of **functions** $\mathcal{H} \in [n] \to [n]$ (where $[n] = \{0, \ldots, n-1\}$) is called **min-wise independent** if for any $X \subseteq [n]$, and for any $x \in X$, where h is chosen uniformly at random in $\mathcal{H}$, we have:

$$Pr_{h \in \mathcal{H}}[\min\{h(X)\} = h(x)] = \frac{1}{|X|}$$

Broder et. al. [22] showed that the size of such families must grow exponentially with n, hence exact construction isn't feasible.

Min hashing is a widely used tool for solving problems in computer science such as estimating similarity [22, 23, 24], rarity [5], transitive closure [4], web page duplicate detection [6, 10, 16, 17], sketching techniques

*Supported by ISF, BSF and Google award.

†Department of Computer Science, Bar-Ilan University, Ramat Gan 52900, Israel. feigeng@cs.biu.ac.il

‡Department of Computer Science, Bar-Ilan University, Ramat Gan 52900, Israel. porately@cs.biu.ac.il

§Department of Computer Science, Bar-Ilan University, Ramat Gan 52900, Israel. shiftaa@cs.biu.ac.il

¶This work is Part of PhD thesis done by Ariel Shiftan in the faculty of Computer Science, Bar-Ilan University, under the supervision of Prof. Ely Porat

[2, 3], greedy list intersection [18], and other data mining problems [7, 13, 15, 8, 9].

One of the key properties of min hashing is that it enables us to sample the universe of the elements being hashed. This is because each element, over the random choice of hash function out of the family, has equal probability of being mapped to the minimal value, regardless of the number of occurrences of the element. Thus, by maintaining the element with the minimal hash value over the input, one can sample the universe. Usually, $k > 1$ element are needed, hence k instances of min hashing schemes are used.

In [2, 3] it was shown that it is better to use one hashing scheme and maintain the k elements with the smallest hash values, in order to get a tighter estimator. Intuitively, this is because the bottom k elements are much more stable than just the minimal, and the number of distinct elements sampled is exactly k (as there are no repetitions).

Similarity estimation of data sets is a fundamental tool in mining data. It is often calculated using the Jaccard similarity coefficient which is defined by $\frac{|A\cap B|}{|A\cup B|}$, where A and B are two data sets. By maintaining the minimal hash value over two sets of data inputs A and B, the probability of getting the same hash value is exactly $\frac{|A\cap B|}{|A\cup B|}$, which equals the Jaccard similarity coefficient, as described in [22, 23, 24, 4].

Cohen et al. [25] proposed an estimator for similarity over sets of data. The idea is to run the above method for calculating Jaccard similarity several times in parallel and take the average. Formally, using notations taken from [5]:

***min-wise* estimator:** Let $h_1(A), h_2(A), ..., h_k(A)$ and $h_1(B), h_2(B), ..., h_k(B)$ be k independent min hash values for the sets A and B, respectively, the similarity is estimated by:

$$\hat{S}(A,B) = \frac{\mid \{l | 1 \leq l \leq k, h_l(A) = h_l(B)\} \mid}{k}$$

For $0 < \epsilon < 1$,$0 < p < 1$, $0 < \tau < 1$, and $k \geq 2\epsilon^{-2}p^{-1}\log\tau^{-1}$

$$\hat{S}(A,B) \in (1 \pm \epsilon)\frac{|A\cap B|}{|A\cup B|} + \epsilon p$$

for a prespecified precision p with success probability at least $1 - \tau$.

For cases where the data inputs are sets, such as in [25], randomly choosing a value for each element is equivalent to hashing each element using min-wise permutation (or function). In more common cases, where each element can appear more than once, a min-wise independent function is needed in order to preserve consistency of hash values of identical elements.

Another estimator is given in [25], where instead of hashing k times, one can hash each element once, and store the k elements with the minimal values.

***k-min-wise* estimator:** Let $h_{1...k}(A)$, $h_{1...k}(B)$ be the set of k minimal hash values for the data inputs A and B respectively, the similarity is estimated by:

$$\hat{S}(A,B) = \frac{\mid h_{1...k}(A) \cap h_{1...k}(B) \cap h_{1...k}(A \cup B) \mid}{k}$$

The precision is roughly the same as for the first estimator.

Indyk in [1] was first to give a construction of a small approximately *min-wise* independent family of hash functions, another construction was proposed in [26].

A family of functions $\mathcal{H} \subseteq [n] \to [n]$ is called **approximately min-wise independent**, or ϵ-*min-wise* independent, if, for any $X \subseteq [n]$, and for any $x \in X$, where h is chosen uniformly at random in $\mathcal{H}$, we have:

$$Pr_{h\in\mathcal{H}}[\min\{h(X)\} = h(x)] = \frac{1}{|X|}(1 \pm \epsilon)$$

where $\epsilon \in (0, 1)$ is the desired error bound, and the number of bits needed to represent each function in Indyk's construction is $O(\log n \log(\frac{1}{\epsilon}))$, matching a lower bound from [27].

Indyk's construction is applicable for estimating similarity in data stream models. In the unbounded data stream model, we consider a stream, in which elements arrive sequentially. Due to the size of the stream, it is only allowed to perform one pass over the data. Furthermore, the storage available is poly-logarithmic in the size of the stream. In the windowed data stream model we consider a predefined size window of size N over the stream, such that all the queries to the stream are related to elements in the current window. Similarly to the unbounded streaming model, we are only allowed one pass over the data and the storage available is poly-logarithmic in the size of the window.

Datar and Muthukrishnan [5] proposed an algorithm for estimating similarity and rarity over data stream windows. The algorithm proposed utilizes Indyk's construction and the *min-wise* estimator given in [25] (presented above, with $k = 2\epsilon^{-2}p^{-1}\log\tau^{-1}$). They showed that in order to have the minimal hash value at any time, the expected number of elements needed to be stored is $O(\log N)$, with high probability, where N is the window size. The time needed by this algorithm to calculate the k hash values for each item is

$O(k \log \frac{1}{\epsilon})$, and the additional processing time per item is $O(k \log \log N)$. The space needed to store the k hash functions is $O(k \log \frac{1}{\epsilon})$ words, and the space for the min values is $O(k \log N)$ words.

1.1 Our Contribution In this paper we give the first known construction of a small approximately *k-min-wise* independent family of hash functions. First, we extend the notion of *min-wise* independent family of hash functions by defining a *k-min-wise* independent family of hash functions. Then, we show the construction of such family. Under this definition, all subsets of size k of any fixed set X have an equal chance to have the minimal hash values among all the elements in X, where the probability is over the random choice of hash function from the family. The formal definition is given in section 2. The number of words needed to represent each function in our construction is $O(k \log \log(\frac{1}{\epsilon}) + \log(\frac{1}{\epsilon}))$, where $\epsilon \in (0, 1)$ is the desired approximation factor. Since, by definition, each subset has equal probability to have the minimal hash values, the k min values have to be independent and thus k lower bounds the space needed for any construction. Hence, our construction, which is only suboptimal by the factor $O(\log \log(\frac{1}{\epsilon}))$, is very efficient. The construction is summarized in the following theorem:

THEOREM 1.1. *There exist constants $c', c'' > 1$ such that for any $|X| < \epsilon n / c'$ and $\epsilon > 0$ any $c''(k \log \log \frac{1}{\epsilon} + \log \frac{1}{\epsilon})$-wise independent family of functions is ϵ-k-min-wise independent.*

This construction is in fact the first applicable for sampling bottom-k sketches out of the universe. Prior constructions, as in [2, 3], were based on assigning a random rank for each element, and therefore did not sample from the universe.

We utilize the construction and propose a simple method for exponential time improvement of *min-wise* based algorithms, such as in [4, 5, 6, 7, 3, 2, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]. These algorithms use $c > 1$ approximately *min-wise* hash functions in order to sample c elements independently from the universe. We propose to replace them with only one approximately *k-min-wise* (for $k = c$) independent function. As the k elements are fully independent, we can get the same precision. The above procedure does not change the algorithm itself, but only the way it samples, and hence it is simple to adapt. We found this to improve exponentially the time complexity and asymptotically the space.

Due to lack of space a full algorithm is only given for similarity, we also discuss briefly rarity and entropy of a random walk. The similarity algorithm utilizes the *k-min-wise* estimator from [25] along with our construction, in order to improve the known results of similarity estimation over data streams in [5]. Our algorithm exponentially reduces the time complexity, and asymptotically the space. Using a sophisticated approach, the time needed for calculating the hash value of each item is **only** $O(\log^2(k \log \log \frac{1}{\epsilon} + \log \frac{1}{\epsilon}))$, and the additional processing time per item is $O(\log k)$. The space needed by the algorithm is $O(k \log \log \frac{1}{\epsilon} + \log \frac{1}{\epsilon})$ for storing the hash functions, and $O(k \log N)$ for the k min hash values. The value of k in defined to be the same as in the original paper [5]. See tables 1 and 2 for comparison of results.

	prev. [5] results	this paper
Hashing time	$k \log \frac{1}{\epsilon}$	$\log^2(k \log \log \frac{1}{\epsilon} + \log \frac{1}{\epsilon})$
Additional time	$k \log \log N$	$\log k$
Functions memory	$k \log \frac{1}{\epsilon}$	$k \log \log \frac{1}{\epsilon} + \log \frac{1}{\epsilon}$
Additional space	$k \log N$	$k \log N$

Table 1: Similarity and rarity algorithms comparison in the windowed data stream model. Time complexity is given per item observed, and space is given in words, in upper bounds.

	prev. [5, 25] results	this paper
Hashing time	$k \log \frac{1}{\epsilon}$	$\log^2(k \log \log \frac{1}{\epsilon} + \log \frac{1}{\epsilon})$
Additional time	k	$\log k$
Functions memory	$k \log \frac{1}{\epsilon}$	$k \log \log \frac{1}{\epsilon} + \log \frac{1}{\epsilon}$
Additional space	k	k

Table 2: Similarity and rarity algorithms comparison in the unbounded data stream model. Time complexity is given per item observed, and space is given in words, in upper bounds.

1.2 Outline The outline of the paper is as follows. In section 2 we define the notion of *k-min-wise* and approximately *k-min-wise* independent families. In section 3 we present a construction of such family. In section 4 we propose two algorithms for similarity and discuss briefly rarity and entropy of a random walk over graphs. Note that one can read section 4 without diving into the details of section 3.

2 Definitions

We will start by giving our definitions of exact and approximately *k-min-wise* independent family of hash functions, which are generalization of *min-wise* inde-

pendent family of hash functions.

DEFINITION 2.1. *k-min-wise independent family of hash functions:*
A family of functions $\mathcal{H} \subseteq [n] \to [n]$ (where $[n] = \{0 \dots n-1\}$) is called k-min-wise independent if for any $X \subseteq [n]$ and for any $Y \subset X$, $|Y| = k$ we have

$$\Pr_{h \in \mathcal{H}} \left[\max_{y \in Y} h(y) < \min_{z \in X-Y} h(z) \right] = \frac{1}{\binom{|X|}{|Y|}},$$

where the function h is chosen uniformly at random from $\mathcal{H}$.

DEFINITION 2.2. *Approximately k-min-wise (ϵ-k-min-wise) independent family of hash functions:*
A family of functions $\mathcal{H} \subseteq [n] \to [n]$ (where $[n] = \{0 \dots n-1\}$) is called ϵ-k-min-wise independent if for any $X \subseteq [n]$ and for any $Y \subset X$, $|Y| = k$ we have

$$\Pr_{h \in \mathcal{H}} \left[\max_{y \in Y} h(y) < \min_{z \in X-Y} h(z) \right] = \frac{1}{\binom{|X|}{|Y|}} (1 \pm \epsilon),$$

where the function h is chosen uniformly at random from $\mathcal{H}$, and $\epsilon \in (0,1)$ is the error bound.

Prior to publishing this work we found these definitions to be closely related to the notion of k-minima-wise independent permutations in [28].

3 The Construction

3.1 Overview of technique In this section we present a construction of a small approximately *k-min-wise* independent family of hash functions $\mathcal{H}$, by extending Indyk's construction of *ϵ-min-wise* independent family of hash functions in [1]. We first define the following probabilistic event Ψ, for any given sets $X \subseteq [n]$ and $Y \subset X$, where $|Y| = k$:

DEFINITION 3.1. $\Psi = [\max_{y \in Y} h(y) < \min_{z \in X-Y} h(z)]$.
Informally:
"all elements of the set Y are hashed to values smaller than the hash values of the elements in $X - Y$"

In order to fulfill definition 2.2, we have to show that the probability of event Ψ, over the random choice of the function, is almost equal (up to multiplicative factor of $1+\epsilon$) for all possible subsets Y of X.
We will start by dividing event Ψ into n disjoint sub-events. We define the sub-event Ψ_i for $i \in [n]$, to be the event where the maximal hash value of the elements in Y is i, and all elements of $X-Y$ have hash values larger than i. Observe that $\Psi = \cup_{i=0}^{n-1} \Psi_i$.
Then we will show that choosing uniformly at random a hash function from our family $\mathcal{H}$ resembles closely choosing uniformly from the family of all functions $[n] \to [n]$.

3.2 Construction in details Let $k = |Y|$, $m = |X|$, $Y \subset X$ and $\epsilon \in (0,1)$. For any $i \in [n]$ let A_i denote the event "the hash values of all the elements of $X - Y$ are larger than i". Note that $\Pr(\Psi_i) = \Pr(A_i \cap \max_{y \in Y} h(y) = i)$. We will calculate the probability of the event Ψ by summing up the events $A_i \cap \max_{y \in Y} h(y) = i$ for all $i \in [n]$.

Define $E_i = \frac{i+1}{n} \cdot m$; notice that E_i is the expected number of elements from X falling into $[0 \dots i]$. Let $\Pr_l[\cdot]$ denote any l-wise independent probability measure over $[n] \to [n]$.

LEMMA 3.1. *Let $\mathcal{H}$ be a family of hash functions.*
If $\Pr_{h \in \mathcal{H}}[\max_{y \in Y} h(y) < \min_{z \in X-Y} h(z)] = \frac{1}{\binom{|X|}{|Y|}} \pm \frac{(\frac{|Y|}{e})^{|Y|}}{|X|^{|Y|}} \epsilon$,
where the function h is chosen uniformly at random from $\mathcal{H}$, then $\mathcal{H}$ is ϵ-k-min-wise independent.

Proof. Since $\binom{|X|}{|Y|} \leq \frac{|X|^{|Y|}}{|Y|!}$, by applying Stirling approximation we get $\frac{1}{\binom{|X|}{|Y|}} \geq \frac{|Y|!}{|X|^{|Y|}} \geq \frac{(\frac{|Y|}{e})^{|Y|}}{|X|^{|Y|}}$. Hence, we get sufficient error bound.

The following two lemmata, lemma 3.2 and lemma 3.3, were taken from [1]:

LEMMA 3.2. *Let i be such that $E_i \leq \frac{l-1}{2e}$ and $l \geq \log(\frac{2}{\epsilon})$. Then $\Pr_l[A_i] = Pr[A_i] \pm 2\epsilon$.*

LEMMA 3.3. *Let l' be even. Then $Pr_{l'}[A_i] \leq 48(\frac{6l'}{E_i})^{(l'-1)/2}$.*

Let $z = k \log\log \frac{1}{\epsilon} + 13k + 7$, $l' = 4\log(\frac{2^z}{\epsilon}) + 2 = 4z + 4\log\frac{1}{\epsilon} + 2$, $t = 12l'$ and $l = et + 1$. We now prove the following lemma:

LEMMA 3.4.

$$100t^k e^k \leq 2^z k^k \quad (\textit{equivalently } \frac{100t^k}{2^z} \leq \frac{k^k}{e^k})$$

Proof. Since

$$(4 + 52 + 28 + 4 + 2)k \log\frac{1}{\epsilon} \leq 2^7 k \log\frac{1}{\epsilon}$$

and, trivially

$$i)\ 4(k\log\log\frac{1}{\epsilon} + 13k + 7) + 4\log\frac{1}{\epsilon} + 2 \leq (4 + 52 + 28 + 4 + 2)k\log\frac{1}{\epsilon}$$

$$ii)\ 2^7 k \log\frac{1}{\epsilon} = 2^{\log\log\frac{1}{\epsilon} + 7} k$$

we have

$$4(k\log\log\frac{1}{\epsilon}+13k+7)+4\log\frac{1}{\epsilon}+2\leq 2^{\log\log\frac{1}{\epsilon}+7}k$$

$$4z+4\log\frac{1}{\epsilon}+2\leq 2^{\frac{z-7}{k}-6}k$$

$$l'\leq 2^{\frac{z-7}{k}-6}k$$

$$64l'\leq 2^{\frac{z-7}{k}}k$$

$$(64l')^k\leq 2^{z-7}k^k$$

$$128(\frac{16}{3}t)^k\leq 2^z k^k$$

Observe that

$$100t^k e^k\leq 128(\frac{16}{3}t)^k$$

and thus

$$100t^k e^k\leq 2^z k^k$$

We use the four above lemmata to prove the following lemma which immediately implies the main theorem.

LEMMA 3.5. *Let $\mathcal{H}$ be any $l+k$ wise independent family of hash functions, where $h\in\mathcal{H}$ is chosen uniformly at random.*

$$\Pr_{l+k}\left[\max_{y\in Y}h(y)<\min_{x\in X-Y}h(x)\right]=\frac{1}{\binom{|X|}{|Y|}}\pm\frac{(\frac{|Y|}{e})^{|Y|}}{|X|^{|Y|}}\cdot 2\epsilon$$

Proof. First, notice that $\Pr[\max_{y\in Y}h(y)=i]=\frac{i^k-(i-1)^k}{n^k}$, as we have i^k possibilities for the elements of Y to be in the range $[1,\ldots,i]$, and in $(i-1)^k$ out of them the largest element is smaller than i.

$$\begin{aligned}
&\Pr_{l+k}\left[\max_{y\in Y}h(y)<\min_{x\in X-Y}h(x)\right]\\
&=\sum_{i=0}^{n-1}\Pr_{l+k}\left[\max_{y\in Y}h(y)=i,\ \min_{x\in X-Y}h(x)>i\right]\\
&=\sum_{i=1}^{n}\frac{i^k-(i-1)^k}{n^k}\Pr_{l+k}[A_i|\textit{values of } h \textit{ for } Y]\\
&=\sum_{i=1}^{\frac{nt}{2m}-1}\frac{i^k-(i-1)^k}{n^k}\Pr_l[A_i]\\
&\quad+\sum_{i=\frac{nt}{2m}}^{n}\frac{i^k-(i-1)^k}{n^k}\Pr_l[A_i]\\
&=P_1+P_2
\end{aligned}$$

where P_1 and P_2 are he first and second terms.

First we handle P_1. By using lemma 3.2 with $\epsilon'=\frac{\epsilon}{2^z}$ we get:

$$P_1=\sum_{i=1}^{\frac{nt}{2m}-1}\frac{i^k-(i-1)^k}{n^k}(\Pr[A_i]\pm 2\epsilon')$$

$$=(\sum_{i=1}^{\frac{nt}{2m}-1}\frac{i^k-(i-1)^k}{n^k}\Pr[A_i])\pm\frac{(\frac{nt}{2m}-1)^k}{n^k}2\epsilon'$$

Since we are comparing the precision to a fully random family of hash functions the second part in the above expression, $\frac{(\frac{nt}{2m}-1)^k}{n^k}2\epsilon'$, bounds the error.

$$\frac{(\frac{nt}{2m}-1)^k}{n^k}2\epsilon'\leq\frac{2\epsilon' t^k}{(2m)^k}=\frac{2\epsilon t^k}{(2m)^k 2^z}$$

Using lemma 3.4 we get

$$\frac{2\epsilon t^k}{(2m)^k 2^z}\leq\frac{(\frac{k}{e})^k}{m^k}\epsilon=\frac{(\frac{|Y|}{e})^{|Y|}}{|X|^{|Y|}}\epsilon$$

which bounds the error as desired.

We now bound the second part, P_2. Using lemma 3.3 we will show that the expression is small enough, specifically smaller than the desired error. Define the event $B_{r,q}$ to be the event where the maximal hash value of Y falls into the interval $[r\ldots q]$. Recall that P_2 is composed of all events where the maximal hash value of Y falls into $[\frac{nt}{2m}\ldots n]$ and the hash values of elements in $X-Y$ are greater. We will group the events into groups of size $\frac{2^i nt}{m}$, for $i\in[-1\ldots\log\frac{m}{2t}]$. Specifically, the first of size $\frac{nt}{2m}$, the next of size $\frac{nt}{m}$ and so on.

Recall that $P_2=\sum_{i=\frac{nt}{2m}}^{n}\frac{i^k-(i-1)^k}{n^k}\Pr_l[A_i]$. Notice that for each interval $[r\ldots q]$, the value $\Pr_l[A_i]$, s.t. $i\in[r\ldots q]$, is bounded by $\Pr_l[A_r]$. Hence,

$$\begin{aligned}
P_2&\leq\sum_{i=\frac{nt}{2m}}^{\frac{nt}{m}}\frac{i^k-(i-1)^k}{n^k}\Pr_l\left[A_{\frac{nt}{2m}}\right]\\
&\quad+\sum_{i=\frac{nt}{m}}^{\frac{2nt}{m}}\frac{i^k-(i-1)^k}{n^k}\Pr_l\left[A_{\frac{nt}{m}}\right]\\
&\quad+\ldots\\
&\quad+\sum_{i=\frac{n}{2}}^{n}\frac{i^k-(i-1)^k}{n^k}\Pr_l\left[A_{\frac{n}{2}}\right]\\
&=\Pr_l\left[A_{\frac{nt}{2m}}\right]\sum_{i=\frac{nt}{2m}}^{\frac{nt}{m}}\frac{i^k-(i-1)^k}{n^k}
\end{aligned}$$

$$+\Pr_l\left[A_{\frac{nt}{m}}\right]\sum_{i=\frac{nt}{m}}^{\frac{2nt}{m}}\frac{i^k-(i-1)^k}{n^k}$$
$$+\ldots$$
$$+Pr_l\left[A_{\frac{n}{2}}\right]\sum_{i=\frac{n}{2}}^{n}\frac{i^k-(i-1)^k}{n^k}$$

Furthermore, by definition $\Pr[B_{r,q}]=\sum_{i=r}^{q}\frac{i^k-(i-1)^k}{n^k}$, and because $\Pr[B_{r,q}]\leq\frac{q^k}{n^k}$ we get:

$$P_2\leq\Pr_l\left[A_{\frac{nt}{2m}}\right]\frac{(\frac{nt}{m})^k}{n^k}+\Pr_l\left[A_{\frac{nt}{m}}\right]\frac{(\frac{2nt}{m})^k}{n^k}+\ldots+Pr_l\left[A_{\frac{n}{2}}\right]\frac{n^k}{n^k}$$

For $x<l$ we can use lemma 3.3 and get:

$$\leq\frac{(\frac{nt}{m})^k}{n^k}\cdot 48(\frac{6x}{\frac{\frac{nt}{2m}\cdot(m)}{n}})^{\frac{x-1}{2}}+\frac{(\frac{2nt}{m})^k}{n^k}\cdot 48(\frac{6x}{\frac{\frac{nt}{m}\cdot(m)}{n}})^{\frac{x-1}{2}}+\ldots+\frac{n^k}{n^k}\cdot 48(\frac{6x}{\frac{\frac{n}{2}\cdot(m)}{n}})^{\frac{x-1}{2}}$$
$$\leq(\frac{t}{m})^k\cdot 48(\frac{12x}{t})^{\frac{x-1}{2}}+(\frac{2t}{m})^k\cdot 48(\frac{6x}{t})^{\frac{x-1}{2}}+\ldots+48(\frac{12x}{m})^{\frac{x-1}{2}}$$
$$\leq(\frac{t}{m})^k\cdot 48(\frac{12x}{t})^{\frac{x-1}{2}}\cdot\left[1+(\frac{1}{2})^{\frac{x-1}{2}-k}+\ldots\right]$$

Notice that $\left[1+(\frac{1}{2})^{\frac{x-1}{2}-k}+\ldots\right]$ is bounded by 2, if $\frac{x-1}{2}-k>1$.
We now choose $x=\frac{l'}{2}$:

$$\leq(\frac{t}{m})^k\cdot 96(\frac{1}{2})^{\log\frac{2^z}{\epsilon}}$$
$$\leq(\frac{t}{m})^k\cdot 96\frac{\epsilon}{2^z}$$

Using lemma 3.4 we get:

$$\leq\frac{(\frac{k}{e})^k}{m^k}\epsilon$$

Putting it all together, we showed

$$P_1=(\sum_{i=1}^{\frac{nt}{2m}-1}\frac{i^k-(i-1)^k}{n^k}\Pr[A_i])\pm\frac{(\frac{k}{e})^k}{m^k}\epsilon$$

and

$$P_2\leq\frac{(\frac{k}{e})^k}{m^k}\epsilon$$

hence,

$$P_1+P_2=\frac{1}{\binom{|X|}{|Y|}}\pm\frac{(\frac{|Y|}{e})^{|Y|}}{|X|^{|Y|}}\cdot 2\epsilon$$

In conclusion, in this section we proved that choosing uniformly at random from $O(k\log\log\frac{1}{\epsilon}+\log\frac{1}{\epsilon})$-wise independent family of functions is approximately *k-min-wise* independent.

4 Applications Over *min-wise* Based Algorithms

One of the possible uses of our construction is similarity estimation of two data streams. As described in the introduction, the problem was studied by [5]. The use of our construction improves exponentially the runtime and asymptotically the space consumption the of current known results. We will now present two algorithms for the problem, in the unbounded and in the windowed data stream models. In addition, we also discuss briefly rarity and entropy of a random walk over graphs. The technique used by the algorithms is general, and can be utilized to improve many *min-wise* based algorithms, such as [4, 5, 6, 7, 3, 2, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18], since most of them handle the min wise functions similarly.

4.1 Similarity Algorithm for the Unbounded Data Stream Model In this section we present a similarity estimation algorithm in the unbounded data stream model. Recall that in this model we consider a stream in which it is only allowed to perform one pass over the data, and the storage available is poly-logarithmic in the size of the stream. Our algorithm uses the *k-min-wise* estimator, with $k=2\epsilon^{-2}p^{-1}\log\tau^{-1}$, as described in the introduction:

$$\hat{S}(A,B)=\frac{|\ h_{1\ldots k}(A)\cap h_{1\ldots k}(B)\cap h_{1\ldots k}(A\cup B)\ |}{k}$$

The value we choose for k and (ϵ,p,τ) are the same as defined in the original paper [5].

First, we randomly choose a function from an approximately *k-min-wise* independent family of hash functions, using the family of $O(k\log\log(\frac{1}{\epsilon})+\log(\frac{1}{\epsilon}))$-degree polynomials over $GF(n)$ (for prime n) . In order

to maintain the lowest k hash values of the elements observed in the stream, we use a binary tree of size k and a buffer of size $c = O(k \log\log \frac{1}{\epsilon} + \log \frac{1}{\epsilon})$ for each stream. We conceptually divide each stream into buckets of size c and insert new arriving elements into the buffer. Whenever the buffer becomes full, we first calculate the hash value of each element in the buffer. Then, we flush the buffer and maintain the lowest k values from both the tree and the buffer inside the tree.

At query time we first flush the buffers. Observe that we have 2 trees, each with the k minimal values of the relevant stream. For the *k-min-wise* estimator, we now take the set of the k lowest hash values among the $2k$ values, $h_{1\ldots k}(A \cup B)$, and intersect it with the two sets of k values, $h_{1\ldots k}(A)$ and $h_{1\ldots k}(B)$. The algorithm result is the size of the intersections divided by k, which provides the similarity estimation.

The space consumption is composed of $O(k)$ words for the trees and $O(k \log\log \frac{1}{\epsilon} + \log \frac{1}{\epsilon})$ for maintaining the hash functions and buffers, where the later bounds the total space consumption. It is known that evaluation of any set of n points of an n-th degree polynomial can be performed in $O(n \log^2 n)$ arithmetic operation [29]. Thus, we can flush and process the buffers in $O(c \log^2 c + c \log k)$ time, which is bounded by $O(\log^2 c)$ per element. Since the two sets are already sorted in the tree the intersection runtime is $O(k)$, therefore the total running time per element is $O(\log^2 c) = O(\log^2(k \log\log \frac{1}{\epsilon} + \log \frac{1}{\epsilon}))$.

4.2 Similarity Algorithm for the Windowed Data Stream Model We now present a similarity estimation algorithm for the windowed data stream model, that improves exponentially the current known results from [5]. It uses the same estimator as in the unbounded data stream algorithm. Recall that in this model we consider a predefined window over the stream, such that all the queries to stream are related to the elements in the current window. Furthermore, it is only allowed to perform one pass over the data and the storage available is poly-logarithmic in the size of the window.

Unlike the algorithm in the unbounded data stream model, the maintenance of the k min hash values is not trivial. Here we need to be able to provide the k min hash values for the current window at any time. In order to provide such hash values we need to save each arriving element and its hash value until k smaller hash values are observed after it. We can drop such element since, obviously, it will not be among the k min hash values at any proceeding window.

LEMMA 4.1. *With high probability over the random choice of approximately k-min-wise function from the family, the expected number of elements needed to be store is $O(k \log N)$, with high probability.*

Proof.
We relate to the worst case, where all elements in the window are different. Observe that in the *min-wise* estimator, as in [5], the probability of the oldest element in the window to have the minimal hash value is $\frac{1}{N}$. This is the probability that the element is needed to be saved. Respectively, the probability for the the second element to be stored is $\frac{1}{N-1}$ and so on. Hence, since the expectation is the sum of probabilities, the expected number of elements needed to be stored is $1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{N} = O(\log N)$. In the *k-min-wise* estimator, when considering the k smallest hash values, the probability of the oldest element in the window to be among the k minimum hash values is $\frac{k}{N}$, for the second is $\frac{k}{N-1}$, and so on. As a result, the expected number of elements to be stored is $O(k \log N)$.

We now present the algorithm. For each stream separately, we store the elements and their arrival time in a linked list, such that the list is ordered by arrival time. In addition, we conceptually divide the stream into buckets of size $c = O(k \log N)$, and use a buffer of that size to store the current bucket. When the buffer becomes full, we perform the following procedure:

Procedure 1

1. Create a binary-tree T.
2. Add the buffer to the beginning of the linked list, such that the first element in the linked list is the last element observed.
3. Traverse the linked list (which is sorted by arrival time). For each element e we do the following:
   (a) If the arrival time of e is out of the window: Remove the element from the list.
   (b) **If** the size of the tree is less than k ($h(e)$ is the hash value of e):
      i. If $h(e)$ is not in T: insert $h(e)$ to T.
      ii. Else: Remove the element from the list.
   (c) **Else if** the size of the tree is k, and $h(e)$ is smaller than the maximum value in T:
      i. If $h(e)$ is not in T: Remove the maximum value in T, and insert $h(e)$ to T.
      ii. Else: Remove the element from the list.
   (d) **Else**: Remove the element from the list.

At query time we first flush the buffers. Observe that we have 2 trees, each with the k minimal values of the relevant stream. For the *k-min-wise* estimator, we now take the set of the k lowest hash values among the $2k$ values, $h_{1...k}(A \cup B)$, and intersect it with the two sets of k values, $h_{1...k}(A)$ and $h_{1...k}(B)$. The algorithm result is the size of the intersections divided by k, which provides the similarity estimation. Note that since we traverse the linked list by arrival time, and remove elements which have hash value larger than the k values in the tree, correctness is preserved.

As for the run time complexity: according to lemma 4.1 the expected size of the linked list is $O(k \log N)$. Whenever we perform *Procedure 1*, we traverse all elements in the linked list. Each of the operations performed is bounded by $\log |T| = \log k$, where $|T|$ is the size of the tree. Therefore, the overall cost of *Procedure 1* is $O(k \log N \log k)$. Since we do it once per $c = O(k \log N)$ elements, the amortized processing time is bounded by $O(\log k)$. For the hash values evaluation — by calculating them in chunks of size $O(k \log\log \frac{1}{\epsilon} + \log \frac{1}{\epsilon})$, it can be done in $O(\log^2(k \log\log \frac{1}{\epsilon} + \log \frac{1}{\epsilon}))$ per element, as described in section 4.1. Since the two sets are already sorted in the tree the intersection runtime is $O(k)$, therefore the total running time per element is $O(\log^2(k \log\log \frac{1}{\epsilon} + \log \frac{1}{\epsilon}))$.

The space consumption is composed of $O(k \log\log \frac{1}{\epsilon} + \log \frac{1}{\epsilon})$ words for maintaining the two hash functions (one per stream). In addition $O(k \log N)$ words needed for maintaining the linked list, by lemma 4.1.

4.3 Rarity Algorithm Similar technique can be applied for rarity estimation. Consider a stream A. $\#\alpha$-rare is defined as the number of elements that appears exactly α times, and #distinct is the number of distinct elements in the stream. The α-rarity of the stream is defined as $R_\alpha = \frac{\#\alpha - rare}{\#distinct}$.

Datar and Muthukrishnan proposed a rarity estimator in [5], using k different approximately *min-wise* functions. They maintain the minimal hash value and the frequency of its corresponding element for each of the functions. We denote the frequency of the minimal hash value for the l-th function with freq(l), the rarity is then estimated as follows:

$$\hat{R_\alpha}(A) = \frac{\mid \{l | 1 \leq l \leq k, freq(l) = \alpha\} \mid}{k}$$

where

$$\hat{R_\alpha}(A) \in (1 \pm \epsilon) R_\alpha(A) + \epsilon p$$

For $0 < \epsilon < 1$,$0 < p < 1$, $0 < \tau < 1$, and $k \geq 2\epsilon^{-2} p^{-1} \log \tau^{-1}$ for a prespecified precision p with success probability at least $1 - \tau$.

Using our construction, one can utilize *k-min-wise* functions to perform only one iteration and improve the overall complexity. This is done by choosing an approximately *k-min-wise* hash function and maintaining the smallest k elements. The exponential time improvement and other improvements using our construction are summarized in tables 1 and 2, for constant values of α.

4.4 Entropy of a Random Walk Over Graphs An algorithm for estimating the entropy of a random walk over graphs was presented in SODA07 [20]. This algorithm uses several approximately *min-wise* hash functions in order to sample a vertex from the universe regardless to the number of occurrences in the stream. In addition, they used a distinct count estimator as either [30, 31] for each sampled vertex. Our technique can be applied in order to reduce the number of iterations. Yet, the overall time complexity is not changed asymptotically as it is dominated by the distinct count estimators.

5 Conclusion and future work

In this paper we defined and gave an efficient time and space construction of approximately *k-min-wise* independent family of hash functions. In particular we proved that choosing uniformly at random from $O(k \log\log \frac{1}{\epsilon} + \log \frac{1}{\epsilon})$-wise independent family of functions is approximately *k-min-wise* independent. Furthermore we utilized our construction to exponentially improve running time and asymptotically the space the of current known results for estimating similarity between two data streams. The use of our construction with the algorithms presented introduces a general and novel technique that can improve both time and space bounds of many algorithms that utilize min-hashing, such as [4, 5, 6, 7, 3, 2, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]. The adjustments needed in order to improve these algorithms are left for future work. Finally we note that as a future work, one might consider defining an alternative family, that uses less amount of randomness and would still be applied to the problems, with some relaxations.

References

[1] Indyk, P.: A small approximately min-wise independent family of hash functions. In: Journal of Algorithms. (1999) 454–456

[2] Cohen, E., Kaplan, H.: Tighter estimation using bottom k sketches. PVLDB **1**(1) (2008) 213–224

[3] Cohen, E., Kaplan, H.: Summarizing data using bottom-k sketches. In: PODC. (2007) 225–234
[4] Cohen, E.: Size-estimation framework with applications to transitive closure and reachability. J. Comput. Syst. Sci. **55**(3) (1997) 441–453
[5] Datar, M., Muthukrishnan, S.: Estimating rarity and similarity over data stream windows. In: In Proceedings of 10th Annual European Symposium on Algorithms, volume 2461 of Lecture Notes in Computer Science. (2002) 323–334
[6] Broder, A.Z.: Identifying and filtering near-duplicate documents. In: COM '00: Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching, London, UK, Springer-Verlag (2000) 1–10
[7] Haveliwala, T.H., Gionis, A., Klein, D., Indyk, P.: Evaluating strategies for similarity search on the web. In: WWW '02: Proceedings of the 11th international conference on World Wide Web, New York, NY, USA, ACM (2002) 432–442
[8] Bachrach, Y., Herbrich, R., Porat, E.: Sketching algorithms for approximating rank correlations in collaborative filtering systems. In Karlgren, J., Tarhio, J., Hyyrö, H., eds.: SPIRE. Volume 5721 of Lecture Notes in Computer Science., Springer (2009) 344–352
[9] Bachrach, Y., Porat, E., Rosenschein, J.S.: Sketching techniques for collaborative filtering. In: The Twenty-First International Joint Conference on Artificial Intelligence (IJCAI 2009), Pasadena, California (2009) 2016–2021
[10] Manku, G.S., Jain, A., Das Sarma, A.: Detecting near-duplicates for web crawling. In: WWW '07: Proceedings of the 16th international conference on World Wide Web, New York, NY, USA, ACM (2007) 141–150
[11] Cormode, G., Muthukrishnan, S.: What's new: finding significant differences in network data streams. IEEE/ACM Trans. Netw. **13**(6) (2005) 1219–1232
[12] Ganguly, S., Garofalakis, M., Rastogi, R.: Processing set expressions over continuous update streams. In: SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, New York, NY, USA, ACM (2003) 265–276
[13] Das, A.S., Datar, M., Garg, A., Rajaram, S.: Google news personalization: scalable online collaborative filtering. In: WWW '07: Proceedings of the 16th international conference on World Wide Web, New York, NY, USA, ACM (2007) 271–280
[14] Gibbons, P.B., Tirthapura, S.: Estimating simple functions on the union of data streams. In: SPAA '01: Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures, New York, NY, USA, ACM (2001) 281–291
[15] Haveliwala, T.H., Gionis, A., Klein, D., Indyk, P.: Evaluating strategies for similarity search on the web. In: WWW '02: Proceedings of the 11th international conference on World Wide Web, New York, NY, USA, ACM (2002) 432–442
[16] Yang, H., Callan, J.: Near-duplicate detection by instance-level constrained clustering. In: SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, New York, NY, USA, ACM (2006) 421–428
[17] Henzinger, M.: Finding near-duplicate web pages: a large-scale evaluation of algorithms. In: SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, New York, NY, USA, ACM (2006) 284–291
[18] Krauthgamer, R., Mehta, A., Raman, V., Rudra, A.: Greedy list intersection. In: IEEE 24th International Conference on Data Engineering, 2008. ICDE 2008. (2008) 1033–1042
[19] Bachrach, Y., Herbrich, R.: Fingerprinting Ratings For Collaborative FilteringTheoretical and Empirical Analysis. (2010)
[20] Chakrabarti, A., Cormode, G., McGregor, A.: A near-optimal algorithm for computing the entropy of a stream. In Bansal, N., Pruhs, K., Stein, C., eds.: SODA, SIAM (2007) 328–335
[21] Mulmuley, K.: Randomized geometric algorithms and pseudo-random generators. In: SFCS '92: Proceedings of the 33rd Annual Symposium on Foundations of Computer Science, Washington, DC, USA, IEEE Computer Society (1992) 90–100
[22] Broder, A.Z., Charikar, M., Frieze, A.M., Mitzenmacher, M.: Min-wise independent permutations (extended abstract). In: STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing, New York, NY, USA, ACM (1998) 327–336
[23] Broder, A.Z.: On the resemblance and containment of documents. In: In Compression and Complexity of Sequences (SEQUENCES97, IEEE Computer Society (1997) 21–29
[24] Broder, A.Z., Glassman, S.C., Manasse, M.S., Zweig, G.: Syntactic clustering of the web. In: Selected papers from the sixth international conference on World Wide Web, Essex, UK, Elsevier Science Publishers Ltd. (1997) 1157–1166
[25] Cohen, E., Datar, M., Fujiwara, S., Gionis, A., Indyk, P., Motwani, R., Ullman, J.D., Yang, C.: Finding interesting associations without support pruning (1999)
[26] Saks, M., Srinivasan, A., Zhou, S., Zuckerman, D.: Low discrepancy sets yield approximate min-wise independent permutation families. In: In Proc. International Workshop on Randomization and Approximation Techniques in Computer Science, Springer (1999) 29–32
[27] Pătraşcu, M., Thorup, M.: On the k-independence required by linear probing and minwise independence. In: Proc. 37th International Colloquium on Automata, Languages and Programming (ICALP). (2010) To appear.
[28] Broder, A.Z., Charikar, M., Mitzenmacher, M.: A derandomization using min-wise independent permutations. In: In Randomization and approximation tech-

niques in computer science, Springer (2003) 15–24
[29] Aho, A.V., Hopcroft, J.E.: The Design and Analysis of Computer Algorithms. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1974)
[30] Bar-Yossef, Z., Jayram, T.S., Kumar, R., Sivakumar, D., Trevisan, L.: Counting distinct elements in a data stream. In: RANDOM '02: Proceedings of the 6th International Workshop on Randomization and Approximation Techniques, London, UK, Springer-Verlag (2002) 1–10
[31] Kane, D.M., Nelson, J., Woodruff, D.P.: An optimal algorithm for the distinct elements problem. In: PODS '10: Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems of data, New York, NY, USA, ACM (2010) 41–52