Sketching for Big Data Recommender Systems Using Fast Pseudo-random Fingerprints

Yoram Bachrach[1] and Ely Porat[2]

[1] Microsoft Research, Cambridge, UK
[2] Bar-Ilan University, Ramat-Gan, Israel

Abstract. A key building block for collaborative filtering recommender systems is finding users with similar consumption patterns. Given access to the full data regarding the items consumed by each user, one can directly compute the similarity between any two users. However, for massive recommender systems such a naive approach requires a high running time and may be intractable in terms of the space required to store the full data. One way to overcome this is using sketching, a technique that represents massive datasets concisely, while still allowing calculating properties of these datasets. Sketching methods maintain very short fingerprints of the item sets of users, which allow approximately computing the similarity between sets of different users.

The state of the art sketch [22] has a very low space complexity, and a recent technique [14] shows how to exponentially speed up the computation time involved in building the fingerprints. Unfortunately, these methods are incompatible, forcing a choice between low running time or a small sketch size. We propose an alternative sketching approach, which achieves both a low space complexity similar to that of [22] and a low time complexity similar to [14]. We empirically evaluate our algorithm using the Netflix dataset. We analyze the running time and the sketch size of our approach and compare them to alternatives. Further, we show that in practice the accuracy achieved by our approach is even better than the accuracy guaranteed by the theoretical bounds, so it suffices to use even shorter fingerprints to obtain high quality results.

1 Introduction

The amount of data generated and processed by computers has shown consistent exponential growth. There are currently over 20 billion webpages on the internet, and major phone companies process tens of gigabytes of call data each day. Analyzing such vast amounts of data requires extremely efficient algorithms, both in terms of running time and storage. This has given rise to the field of massive datasets processing. We focus on massive recommender systems, which provide users with recommendations for items that they are likely to find interesting, such as music, videos, or web pages. These systems keep a profile of each user and compare it to reference characteristics. One approach is *collaborative filtering* (CF), where the stored information is the items consumed or rated by the user in the past. CF systems predict whether an item is likely to interest the target user by seeking users who share similar rating patterns with the target user and then using the ratings from those like-minded users to generate a prediction for the target user. Various user similarity measures have been proposed, the

F.V. Fomin et al. (Eds.): ICALP 2013, Part II, LNCS 7966, pp. 459–471, 2013.
© Springer-Verlag Berlin Heidelberg 2013

most prominent one being the Jaccard similarity [8,4]. A naive approach, which maintains the entire dataset of the items examined by each user and their ratings and directly computes the similarity between any two users, may not be tractable for big data applications, both in terms of space and time complexity. A recommender system may have tens of millions of users[1], and may need to handle a possible set of billions of items[2]. A tractable alternative requires representing knowledge about users concisely while still allowing inference on relations between users (e.g. user similarity). One method for concisely representing knowledge in such settings is sketching (also known as fingerprinting) [2,20,9,1]. Such methods store *fingerprints*, which are concise descriptions of the dataset, and can be though of as an extreme lossy compression method. These fingerprints are extremely short, far more concise than traditional compression techniques would achieve. On the other hand, as opposed to compression techniques, they do not allow a full reconstruction of the original data (even approximately), but rather only allow inferring very specific properties of the original dataset. Sketching allows keeping short fingerprints of the item sets of users in recommender systems, that can still be used to approximately compute similarity measures between any two users [5,4,3]. Most such sketches use random hashes [2,20,9,1,5].

Our Contribution. The state of the art sketch in terms of space complexity applies many random hashes to build a fingerprint [22], and stores only a single bit per hash. A drawback of this approach is its high running time, caused by the many applications of hashes to elements in the dataset. The state of the art method in terms of running time is [14], which exponentially speeds up the computation time for building the fingerprints. Unfortunately, the currently best sketching techniques in terms of space complexity [22] and time complexity [14] are mutually incompatible, forcing a choice between reducing space or reducing runtime. Also, the low time complexity method [14] is tailored for computing Jaccard similarity between users, but is unsuitable for other similarity measures, such as the more sensitive rank correlation similarity measures [4]. We propose an alternative general sketching approach, which achieves both a low space complexity similar to that of [22] and a low time complexity similar to [14]. Our sketch uses random hashing and has a similar space complexity to [22], storing a single bit per hash, thus outperforming previous approaches such as [2,13,5] in *space* complexity. Similarly to [14], we get an *exponential* speedup in one factor of the *computation time*. Our discussion focuses on Jaccard similarity [5], but our approach is more general than [14], capturing other fingerprints, such as frequency moments [2], L_p sketches [17], rarity [13] and rank correlations [4]. Our sketch "ties" hashes in a novel way, allowing an *exponential* runtime speedup while storing only a single bit per hash. We also make an **empirical contribution** and evaluate our method using the Netflix [6] dataset. We analyze the running time and space complexity of our sketch, comparing it to the above state of the

[1] For example, Netflix is a famous provider of on-demand internet streaming media and employs a recommender system with over 20 million users. Our empirical evaluation is based on the dataset released by Netflix in a challenge to improve their algorithms [6].

[2] An example is a recommender system for webpages. Each such a webpage is a potential information item, so there are billions of such items that can potentially be recommended.

art methods. We show that it in practice the accuracy of our sketch is higher than the theoretical bounds, so even shorter sketches achieve high quality results.

Related Work. Recent work [5,4,3] already suggests sketching for recommender systems. Rather than storing the full lists of items consumed by each user and their ratings, they only store fingerprints of the information for each user, designed so that given the fingerprints of any two users one can accurately estimate the similarity between them. These fingerprints are constructed using min-wise independent families of hash functions, *MWIFs* for short. MWIFs were introduced in [23,8], and are useful in many applications as they resemble random permutations. Much of the research on sketching focused on reducing *space complexity* while accurately computing data stream properties, but much less attention was given to *time complexity*. Recent work that *does* focus on running time as well as space is [18,19] and [12] which propose low runtime sketches for locally sensitive hashing under the l_2 norm. Many streaming algorithms apply *many* hashes to each element in a very long stream of elements, leading to a high and sometimes intractable computation time. Similarly to [14], our method achieves an *exponential* speedup in one factor of the *computation time* for constructing fingerprints of massive data streams. The heart of the method lies in using a specific family of pseudo-random hashes shown to be approximately-MWIF [16], and for which we can quickly locate the hashes resulting in a small value of an element under the hash. Similarly to [24] we use the fact that family members are pairwise independent between themselves. Whereas previous models examine only one hash at a time, we read and process "chunks" of hashes to find important elements in the chunk, exploiting the chunk's structure to significantly speed up computation. We show that our technique is compatible with storing a *single* bit rather than the full element IDs, improving the fingerprint size, similarly to [22].

Improving Time and Space Complexity. Rather than storing the full stream of b items of a universe $[u]$, sketching methods only store a fingerprint of the stream. Any sketch achieves an estimate that is "probably approximately correct", i.e. with high probability the estimation error is small. Thus the size of the fingerprint and the time required to compute it depend on the accuracy of the method ϵ and its confidence δ. The accuracy ϵ is the allowed error (the difference between the estimated similarity and the true similarity), and the confidence δ is the maximal allowed probability of obtaining an estimate with a "large error", higher than the allowed error ϵ. Similarly to [22] we store a single bit per hash function, which results in a fingerprint of length $O(\frac{\ln \frac{1}{\delta}}{\epsilon^2})$ bits, rather than $O(\log u \cdot \frac{\ln \frac{1}{\delta}}{\epsilon^2})$ bits required by previous approaches such as [14]. On the other hand, similarly to [14], rather than computing the fingerprint of a stream of b items in time of $O(\frac{b \ln \frac{1}{\delta}}{\epsilon^2})$ as required by [22], we can compute it in time $O(b \cdot \log \frac{1}{\delta} \cdot \log \frac{1}{\epsilon})$, achieving an exponential speedup for the fingerprint construction. In addition to the theoretical guarantees, in Section 3 we evaluate our approach on the Netflix dataset, contrasting it with previous approaches in terms of time and space complexity. We also show that a high accuracy can be obtained even for very small fingerprints. Our theoretical results relate the required storage to the accuracy of the Jaccard similarity estimates, but only

provide an upper bound regarding the storage required; we show that in practice the storage required to achieve a good accuracy can be much lower.

Preliminaries. Let H be a family of hashes over source X and target Y, so $h \in H$ is a function $h : X \to Y$, where Y is ordered. We say H is min-wise independent if when randomly choosing $h \in H$, for any subset $C \subseteq X$, any $x \in C$ has an equal probability to be minimal after applying h.

Definition 1. *H is min-wise independent (MWIF), if for all $C \subseteq X$, for any $x \in C$, $Pr_{h \in H}[h(x) = min_{a \in C} h(a)] = \frac{1}{|C|}$.*

Definition 2. *H is a γ-approximately min-wise independent (γ-MWIF), if for all $C \subseteq X$, for any $x \in C$: $\left| Pr_{h \in H}[h(x) = min_{a \in C} h(a)] - \frac{1}{|C|} \right| \leq \frac{\gamma}{|C|}$*

Definition 3. *H is k-wise independent, if for all $x_1, x_2, \ldots, x_k \subseteq X$, $y_1, y_2, \ldots, y_k \subseteq Y$, $Pr_{h \in H}[(h(x_1) = y_1) \wedge \ldots \wedge (h(x_k) = y_k)] = \frac{1}{|Y|^k}$*

Pseudo-random Family of Hashes. We describe our hashes. Given a universe of item IDs $[u]$, consider a prime p, such that $p > u$. Consider taking random coefficients for a d-degree polynomial in $\mathbb{Z}_p$. Let $a_0, a_1, \ldots, a_d \in [p]$ be chosen uniformly at random from $[p]$, and the polynomial in $\mathbb{Z}_p$: $f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_d x^d$. Denote by F_d all d-degree polynomials in $\mathbb{Z}_p$ with coefficients in $\mathbb{Z}_p$. Our method chooses members of this family uniformly at random. Indyk [16] shows that choosing a function f from F_d uniformly at random results in F_d being a γ-MWIF for $d = O(\log \frac{1}{\gamma})$. Randomly choosing $a_0, \ldots, a_d$ is equivalent to choosing a member of F_d uniformly at random, so $f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_d x^d$ is a hash chosen at random from the γ-MWIF F_d. Similarly, let $b_0, b_1, \ldots, b_d \in [p]$ be chosen uniformly at random from $[p]$, and $g(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_d x^d$, also a hash chosen at random from the γ-MWIF F_d. Consider the hashes $h_0(x) = f(x), h_1(x) = f(x) + g(x), h_2(x) = f(x) + 2g(x), \ldots, h_i(x) = f(x) + ig(x), \ldots, h_{k-1}(x) = f(x) + (k-1)g(x)$. We call the construction procedure for $f(x), g(x)$ the *base random construction*, and the construction of h_i the *composition construction*. We prove properties of such hashes. We denote the probability of an event E when the hash h is constructed by choosing f, g using the base random construction and composing $h(x) = f(x) + i \cdot g(x)$ (for some $i \in [p]$) as $Pr_h(E)$. These constructions are similar to the constructions used in [21], however our hashes are min-wise independent and pair-wise independent between themselves.

Lemma 1 (Uniform Minimal Values). *Let f, g be constructed using the base construction, using $d = O(\log \frac{1}{\gamma})$. For any $z \in [u]$, any $X \subseteq [u]$ and any value i used to compose $h(x) = f(x) + i \cdot g(x)$: $Pr_h[h(z) \leq min_{y \in X}(h(y)] = (1 \pm \gamma)\frac{1}{|X|}$. Proof in full version.*

Lemma 2 (Pairwise Interaction). *Let f, g be constructed using the base construction for $d = O(\log \frac{1}{\gamma})$. For all $x_1, x_2 \in [u]$ and $X_1, X_2 \subseteq [u]$, and all $i \neq j$ used to compose $h_i(x) = f(x) + i \cdot g(x)$ and $h_j(x) = f(x) + j \cdot g(x)$: $Pr_{f,g \in F_d}[(h_i(x_1) \leq min_{y \in X_1} h_i(y)) \wedge (h_j(x_2) \leq min_{y \in X_2} h_i(y))] = (1 \pm \gamma)^2 \frac{1}{|X_1| \cdot |X_2|}$. Proof in full version.*

2 Collaborative Filtering Using Pseudo-random Fingerprints

Collaborative filtering systems provide a target user recommendations for items based on the consumption patterns of other users. Many such systems rank users by their *similarity* to the target user in their past consumption, then find items many such users have consumed by the target user has not yet consumed and recommend them to the target user [26,27,11]. We focus on estimating the similarity between users (see [28] for a survey of methods for generating recommendations based on similarity information). A common measure of similarity between two users is Jaccard similarity. Our item universe consists of the IDs of all items users may consume, $[u] = \{1, 2, 3, \ldots, u\}$. There are u different items in the universe, but each user only examined some of these. Consider one user who has consumed the items $C_1 \subseteq [u]$ in the past, and another user who has consumed the items $C_2 \subseteq [u]$. The Jaccard similarity between the two users is $J_{1,2} = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$. Several fingerprinting methods were proposed for approximating relations between massive datasets, including the Jaccard similarity [8]. We use hashes defined earlier to exponentially speed up such computations. We use pseudo-random effects, so we must relax the MWIF requirement to a pairwise independence requirement (2-wise independence). We briefly review earlier sketches for estimating Jaccard similarity. Consider a hash $h \in H$ chosen from a MWIF H (or a γ-MWIF). We apply h on all elements C_1 and examine the minimal integer we get, $m_1^h = \arg\min_{x \in C_1} h(x)$. We do the same to C_2 and examine $m_2^h = \arg\min_{x \in C_2} h(x)$. Jaccard similarity sketches are based on computing the probability that $m_1 = m_2$: $Pr_{h \in H}[m_1^h = m_2^h] = Pr_{h \in H}[\arg\min_{x \in C_1} h(x) = \arg\min_{x \in C_2} h(x)]$. Theorems 1 and 2 are proven in [7,8] regarding a hash h chosen uniformly at random from a MWIF H and γ-MWIF, correspondingly.[3]

Theorem 1 (Jaccard / Collision Probability (MWIF)). $Pr_{h \in H}[m_i^h = m_j^h] = J_{i,j}$.

Theorem 2 (Jaccard / Collision Probability (γ-MWIF)). $|Pr_{h \in H}[m_i^h = m_j^h] - J_{i,j}| \leq \gamma$.

Rather than storing the full C_i's, previous approaches [7,8] store their fingerprints. Given k hashes $h_1, \ldots, h_k$ randomly chosen from an γ-MWIF, we can store $m_i^{h_1}, \ldots, m_i^{h_k}$. Given C_i, C_j, for any $x \in [k]$, the probability that $m_i^{h_x} = m_j^{h_x}$ is $J_{i,j} \pm \gamma$. A hash h_x where we have $m_i^{h_x} = m_j^{h_x}$ is a *hash collision*. We can estimate $J_{i,j}$ by counting the proportion of collision hashes out of all the chosen hashes. In this approach, the fingerprint contains k item identities in $[u]$, since for any x, $m_i^{h_x}$ is in $[u]$. Thus, the fingerprint requires $k \log u$ bits. To achieve an accuracy ϵ and confidence δ, such approaches require $k = O(\frac{\ln \frac{1}{\delta}}{\epsilon^2})$.

Our General Approach. We use a "block fingerprint" that estimates $J = J_{i,j}$ with accuracy ϵ and confidence of $\frac{7}{8}$. It stores a *single bit* per hash (where many previous approaches store $\log u$ bits per hash). Later we show how to get a given accuracy ϵ and confidence δ, by combining several such blocks. To get a single bit per hash, we use a hash mapping elements in $[u]$ to a single bit — $\phi : [u] \to \{0, 1\}$, taken from

[3] The full version includes a proof for Theorem 1.

a pairwise independent family (PWIF for short) of hashes. Rather than using $m_i^h = \arg\min_{x \in C_1} h(x)$ we use $m_i^{\phi,h} = \phi(\arg\min_{x \in C_1} h(x))$. Storing $m_i^{\phi,h}$ rather than m_i^{ϕ} shortens the sketch by a $\log u$ factor.

Theorem 3. $Pr_{h \in H}[m_i^{\phi,h} = m_j^{\phi,h}] = \frac{J_{i,j}}{2} + \frac{1}{2} \pm \frac{\gamma}{2}$.

Proof. $Pr_{h \in H, \phi \in H'}[m_i^{\phi,h} = m_j^{\phi,h}] = Pr[m_i^{\phi,h} = m_j^{\phi,h} | m_i^h = m_j^h] \cdot Pr_{h \in H}[m_i^h = m_j^h] + Pr[m_i^{\phi,h} = m_j^{\phi,h} | m_i^h \neq m_j^h] \cdot Pr_{h \in H}[m_i^h \neq m_j^h] = 1 \cdot Pr_{h \in H}[m_i^h = m_j^h] + \frac{1}{2} \cdot (1 - Pr_{h \in H}[m_i^h = m_j^h]) = \frac{1 + J_{i,j} \pm \gamma}{2}$

The purpose of the fingerprint block is to estimate of $J_{i,j}$ with accuracy ϵ. We use $k = \frac{8.02}{\epsilon^2}$ hashes. Denote $\alpha = \frac{2^{10}-1}{2^{10}}$, and let $\gamma = (1-\alpha) \cdot \epsilon = \frac{1}{2^{10}}\epsilon$. We construct a γ-MWIF[4]. To construct the family, consider choosing $a_0, \ldots, a_d$ and $b_0, b_1, \ldots, b_d$ uniformly at random from $[p]$, constructing the polynomials $f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_d x^d$, $g(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_d x^d$, and using the k hashes $h_i(x) = f(x) + i g(x)$, where $i \in \{0, 1, \ldots, k-1\}$.[5]. We also use a hash $\phi : [u] \to \{0,1\}$ chosen from the PWIF of such hashes. We say there is a collision on h_l if $m_i^{\phi,h_l} = m_j^{\phi,h_l}$, and denote the random variable Z_l where $Z_l = 1$ if there is a collision on h_l for users i, j and $Z_l = 0$ if there is no such collision. $Z_l = 1$ with probability $\frac{1}{2} + \frac{J}{2} \pm \frac{\gamma}{2}$ and $Z_l = 0$ with probability $\frac{1}{2} - \frac{J}{2} \pm \frac{\gamma}{2}$. Thus $E(Z_l) = \frac{1}{2} + \frac{J}{2} \pm \frac{\gamma}{2}$. Denote $X_l = 2Z_l - 1$. $E(X_l) = 2E(Z_l) - 1 = J \pm \gamma$. X_l can take two values, -1 when $Z_l = 0$, and 1 when $Z_l = 1$. Thus X_l^2 always takes the value of 1, so $E(X_l^2) = 1$. Consider $X = \sum_{l=1}^{k} X_l$, and take $Y = \hat{J} = \frac{X}{k}$ as an estimator for J. We show that for the above k, Y is accurate up to ϵ with probability at least $\frac{7}{8}$.

Theorem 4 (Simple Estimator). $Pr(|Y - J| \leq \epsilon) \geq \frac{7}{8}$. *Proof in full version.*

Due to Theorem 4, we approximate J with accuracy ϵ and confidence $\frac{7}{8}$ using a "block fingerprint" for C_i, composed of $m_i^{h_1,\phi_1}, \ldots, m_i^{h_k,\phi_k}$, where $h_1, \ldots, h_k$ are random members of a γ-MWIF and $\phi_1, \ldots, \phi_k$ are chosen from the PWIF of hashes $\phi : [u] \to \{0,1\}$. It suffices to take $k = O(\frac{1}{\epsilon^2})$ to achieve this. Constructing each h_i can be done by choosing f, g using the base random construction and composing $h_i(x) = f(x) + i \cdot g(x)$. The base random construction chooses f, g uniformly at random from F_d, the family of d-degree polynomials in $\mathbb{Z}_p$, where $d = O(\log \frac{1}{\epsilon})$. This achieves a γ-MWIF where $\gamma = (1-\alpha) \cdot \epsilon = \frac{1}{2^{10}}\epsilon$.

Achieving a Desired Confidence. We combine several *independent* fingerprints to increase the confidence to a level δ. Earlier this section we proposed a fingerprint of length k to get a confidence of $\frac{7}{8}$. Consider taking m fingerprints for each stream, each of length k. Given two streams, i, j, we have m pairs of fingerprints, each approximating J with accuracy ϵ, and confidence $\frac{7}{8}$. Denote the obtained estimators by $\hat{J}_1, \hat{J}_2, \ldots, \hat{J}_m$, and the *median* of these values by $\hat{J}$. Consider using $m > \frac{32}{9} \ln \frac{1}{\delta}$ "blocks".

[4] The accuracy γ is stronger than the ϵ required of the full fingerprint, for reasons examined later.

[5] This is similar to [21], but here the hashes are MWIF and pair-wise independent between themselves.

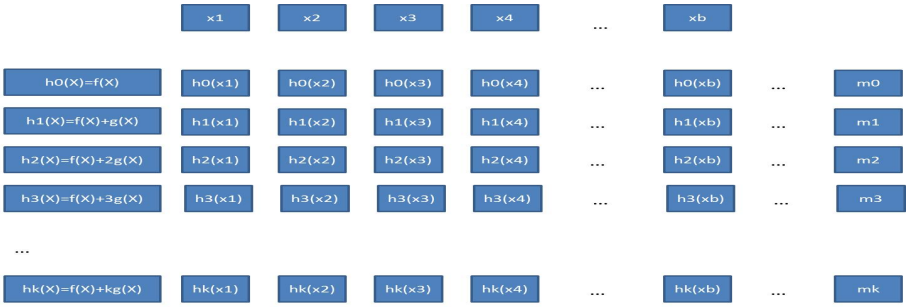

Fig. 1. A fingerprint "chunk" for a stream

Theorem 5 (Median Estimator). $Pr(|\hat{J} - J| \leq \epsilon) \geq 1 - \delta$. *Proof in full version.*

By Theorem 5, to get $|\hat{J} - J| \leq \epsilon$ it suffices to take $m > \frac{32}{9} \ln \frac{1}{\delta}$ blocks, each with $k = \frac{8.02}{\epsilon^2}$ hashes, or $\frac{32}{9} \ln \frac{1}{\delta} \cdot \frac{8.02}{\epsilon^2} \leq \frac{28.45 \ln \frac{1}{\delta}}{\epsilon^2}$ hashes in total. Thus, we use $O(\frac{\ln \frac{1}{\delta}}{\epsilon^2})$ hashes.

2.1 Fast Fingerprint Computation

We discuss speeding up the fingerprint computation for a set of b items $X = \{x_1, \ldots, x_b\}$ where $x_i \in [u]$. The fingerprint has m "block fingerprints", with block r constructed using k hashes $h_1^r, \ldots, h_k^r$, built using $2 \cdot d$ random coefficients in $\mathbb{Z}_p$. The i'th location in the block is the minimal item in X under h_i: $m_i = \arg\min_{x \in X} h_i(x)$, which is then hashed through a hash ϕ mapping elements in $[u]$ to a single bit. We show how to quickly compute the block fingerprint $(m_1, \ldots, m_k)$. A naive way to do this is applying $k \cdot b$ hashes to compute $h_i(x_j)$ for $i \in [k], j \in [b]$. The values $h_i(x_i)$ where $i \in [k], j \in [b]$ form a matrix, where row i has the values $(h_i(x_1), \ldots, h_i(x_b))$, shown in Figure 1.

Once all $h_i(x_j)$ values are computed for $i \in [k], j \in [b]$, for each row i we check for which column j the row's minimal value occurs, and store $m_i = x_j$. Computing the fingerprint requires finding the minimal value across the rows (and the value x_j for the column j where this minimal value occurs). To speed up the process, we use a method similar to [25] as a building block. Recall the hashes h_i were defined as $h_i(x) = f(x) + ig(x)$ where $f(x), g(x)$ are d-degree polynomials with random coefficients in $\mathbb{Z}_p$. Our algorithm is based on a procedure that gets a value $x \in [u]$ and a threshold t, and returns all elements in $(h_0(x), h_1(x), \ldots, h_{k-1}(x))$ which are smaller than t, as well as their locations. Formally, the method returns the index list $I_t = \{i | h_i(x) \leq t\}$ and the value list $V_t = \{h_i(x) | i \in I_t\}$ (note these are lists, so the j'th location in V_t, $V_t[j]$, contains $h_{I_t[j]}(x)$). We call this the *column procedure*, and denote by $pr - small - loc(f(x), g(x), k, x, t)$ the function that returns I_t, and by $pr - small - val(f(x), g(x), k, x, t)$ the function that returns V_t . We describe an implementation of these operations later in this section, with a running time of $O(\log k + |I_t|)$, rather than the naive algorithm which evaluates $O(k)$ hashes. Thus, this

procedure quickly finds small elements across columns (by "small" we mean smaller than t). Our algorithm keeps a bound for the minimal value for each row. It goes through the columns, finding the small values in each, and updates the row bounds where these occur.

block-update $((x_1, \ldots, x_b), f(x), g(x), k, t)$:

1. Let $m_i = \infty$ for $i \in [k]$ and let $p_i = 0$ for $i \in [k]$
2. For $j = 1$ to b:
   (a) Let $I_t = pr - small - val(f(x), g(x), k, x_j, t)$
   (b) Let $V_t = pr - small - loc(f(x), g(x), k, x_j, t)$
   (c) For $y \in I_t$: // Indices of the small elements
      i. If $m_{I_t[y]} > V_t[y]$ // Update to row x required
         A. $m_{I_t[y]} = V_t[y], p_{I_t[y]} = x_j$

If our method updates m_i, p_i for row i, m_i indeed contains the minimal value in that row, and p_i the column where this minimal value occurs, since if even a single update occurred then the row indeed contains an item that is smaller than t, so the minimal item in that row is smaller than t and an update would occur for that item. On the other hand, if all the items in a row are bigger than t, an update would not occur for that row. The running time of the column procedure is $O(\log k + |I_t|)$, which is a random variable, that depends on the number of elements returned for that column, $|I_t|$. Denote by L_j the number of elements returned for column j (i.e. $|I_t|$ for column j). Since we have b columns, the running time of the block update is $O(b \log k) + O(\sum_{j=1}^{b} L_j)$. The total number of returned elements is $\sum_{j=1}^{b} L_j$, which is the total number of elements that are smaller than t. We denote by $Y_t = \sum_{j=1}^{b} L_j$ the random variable which is the number of all elements in the block that are smaller than t. The running time of our block update is thus $O(b \log k + Y_t)$. The random variable Y_t depends on t, since the smaller t is the less elements are returned and the faster the column procedure runs. On the other hand, we only update rows whose minimal value is below t, so if t is too low we have a high probability of having rows which are not updated correctly. A certain compromise t value combines a good running time of the block update with a good probability of correctly computing the values for all the rows.

Theorem 6. *Given the threshold* $t = \frac{12 \cdot p \cdot l'}{b}$, *where* $l' = 80 + 2 \log \frac{1}{\epsilon}$ *(so* $l' = O(\log \frac{1}{\epsilon})$*), the runtime of the* $block - update$ *procedure is* $O(b \log \frac{1}{\epsilon} + \frac{1}{\epsilon^2} \log \frac{1}{\epsilon})$. *Proof in full version.*

Computing Minimal Elements in the Series. We recursively implement $pr - small - loc(f(x), g(x), k, x, t)$ and $pr - small - val(f(x), g(x), k, x, t)$, the procedures for computing V_t and I_t. The hashes h_i were defined as $h_i(x) = f(x) + ig(x)$ where $f(x), g(x)$ are d-degree polynomials with random coefficients in $\mathbb{Z}_p$. Consider $x \in \mathbb{Z}_p$ for which we seek all the values (and indices) in $(h_0(x), h_2(x), \ldots, h_{k-1}(x))$ smaller than t. Given x, we evaluate $f(x), g(x)$ in time $O(d) = O(\log \frac{1}{\gamma})$[6], and denote $a =$

[6] Using multipoint evaluation we can calculate it in amortized time $O(\log^2 \log \frac{1}{\gamma})$. We can use other constructions for d-wise independent which can be evaluated in $O(1)$ time but use more space.

$f(x) \in \mathbb{Z}_p$ and $b = g(x) \in \mathbb{Z}_p$. Thus, we seek all values in $\{a \mod p, (a+b) \mod p, (a+2b) \mod p, \ldots, (a+(k-1)b) \mod p\}$ smaller than t, and the indices i where they occur. Consider the series $S = (s_1, \ldots, s_k)$ where $s_i = (a+ib) \mod p$ and $i = \{0, 1, \ldots, k-1\}$. We denote the arithmetic series $a + bi \mod p$ for $i \in \{0, 1, \ldots, k-1\}$ as $S(a, b, k, p)$, so under this notation $S = S(a, b, k, p)$. Given a value we can find the index where it occurs, and vice versa. To get the value for index i, we compute $(a + ib) \mod p$. To get the index i where value v occurs, we solve $v = a + ib$ in $\mathbb{Z}_p$ (i.e. $i = \frac{v-a}{b} \mod p$). This can be done in $O(\log p)$ time using Euclid's algorithm. We compute b^{-1} in $\mathbb{Z}_p$ *once* to transform *all* values to generating indices. We call a location i where $s_i < s_{i-1}$ a *flip location*. The first index is a flip location if $a - b \mod p > a$. First, consider $b < \frac{p}{2}$. If s_i is a flip location, we have $s_{i-1} < p$ but $s_{i-1} + b > p$, so $s_i < b$. As $b < \frac{p}{2}$ there's at least one location that is *not* a flip location between any two flip locations. Given $S = S(a, b, k, p)$, denote by $f(S)$ the flip locations in S. The flip locations of S are $f(S)$. Denote $f_0(S) = f(S)$, and by $f_i(S)$ elements occurring i places after the closest flip location. Lemmas 3 and 4 are proven in the full version.

Lemma 3 (Flip Locations Are Small). *If $b < \frac{p}{2}$, at most $\frac{k}{2}$ elements are flip locations, and all elements that are smaller than b are flip locations.*

Lemma 4 (Element Comparison). *If $b < \frac{p}{2}$, $x \in f_i(S), y \in f_j(S)$ for $i > j$, then $x > y$.*

The first flip location is $\lceil \frac{p-a}{b} \rceil$, as to exceed p we add $b \lceil \frac{p-a}{b} \rceil$ times. There are $\lfloor \frac{a+bk}{p} \rfloor$ flip locations. Denote the first flip location as $j = \lceil \frac{p-a}{b} \rceil$, with value $a' = (a + jb) \mod p$. Denote $b' = (b - p) \mod b$ and the number of flip locations as $k' = \lfloor \frac{(a+bk)}{p} \rfloor$. The flip locations have an arithmetic progression [25].

Lemma 5 (Flip Locations Arithmetic Progression). *The flip locations of $S = S(a, b, k, p)$ are also an arithmetic progression $S' = (a', b', k', b)$.*

Using these lemmas, we search for the elements smaller than t by examining the flip locations series in recursion. If case $b < t$, given $q = \lceil t \rceil b$, due to Lemma 4 $f(S), f_1(S), \ldots f_{q-1}(S)$ are smaller then t, and all of their elements must be returned. We must also scan $f_q(S)$ and also return all the elements of $f_q(S)$ which are smaller then t. This additional scan requires $O(|f_q(S)|)$ time $|f_q(S)| \leq |f(S)|$. Thus the case of $b < t$ examines $O(|I_t|)$ elements. By Lemma 3, if $b > t$, all non-flip locations are bigger than b and thus bigger than t, so we need only consider the flip-locations as candidates. Using Lemma 5 we scan the flip locations recursively, examining the arithmetic series of the flip locations. If at most half of the elements in each recursion are flip locations, this gives a logarithmic running time, but if b is high, more than half the elements are flip locations. When $b > \frac{p}{2}$ we examine the same flip-location series S', in reverse order. The first element in the reversed series is the last element of the current series, and rather than progressing in steps of b, we progress in steps of $p - b$. Thus we obtain the same elements, but in reverse order. In this reversed series, at most half the elements are flip locations. The procedure below implements our method. It finds elements smaller then t in time $O(\log k) = O(\log \frac{1}{\epsilon} + |I_t|)$ where $|I_t|$ is the number of

such values. Given the returned indices, we get the values in them. We use the same b for all $|I_t|$, so this can be done in time $O(c \log c + |I_t|)$ (usually c is a constant).

ps-min(a, b, p, k, t) :

1. if $b < t$:
   (a) $V_t = []$; if $a < t$ then $V_t = V_t + [a + ib$ for i in range $(\lceil \frac{t-a}{b} \rceil)]$
   (b) $j = \lceil \frac{p-a}{b} \rceil$ // First flip (excluding first location)
   (c) while $j < k$:
      i. $v = (a + jb) \mod p$
      ii. while $j < k$ and $v < t$:
         A. V_t.append(v); $j = j + 1$; $v = v + b$
      iii. $j = j + \lceil \frac{p-v}{b} \rceil$ //next flip location
      iv. return list1
   (d) if $b > \frac{p}{2}$ then return $f((a + (k-1) \cdot b) \mod p, p - b, p, k, t)$
   (e) $j = \lceil \frac{p-a}{b} \rceil$; $new_k = \lfloor \frac{a+bk}{p} \rfloor$
   (f) if $a < b$ then $j = 0$ and $new_k = new_k + 1$// get first flip location
   (g) return $f((a + jb) \mod p, -p \mod b, b, new_k, t)$

3 Empirical Analysis

We empirically evaluated our sketch using the Netflix dataset [6]. This is a movie ratings dataset, with 100 million movie ratings, provided by roughly half a million users on a collection of 17,000 movies. As there are 100 million ratings, even this meduim-sized dataset is difficult to fit in memory[7], so a massive recommender systems dataset certainly cannot fit in the main memory, making sketching necessary to handle such datasets [5]. The state of the art space complexity is achieved using the sketching technique of [22]. Consider using it to estimate Jaccard similarity, with a reasonable accuracy of $\epsilon = 0.01$ and confidence level of $\delta = 0.001$. The approach of [22] applies roughly 100,000 hash functions for each entry. Each hash computation requires 20 multiplication operations, and as there are 100 million entries in the dataset, sketching the entire dataset requires over than $2 \cdot 10^{14}$ multiplications. This takes more than a day to run on a powerful machine. On the other hand, although the approach of [14] allows a much shorter running time (less than an hour), it requires sacrificing the low space achieved by the method of [22]. We first compare our approach with [22,14] in terms of the running time. Figure 2 shows the running time for generating a fingerprint for a target with 1,000 items, both under our method (FPRF - Fast Pseudo Random Fingerprints), and under the sketch of [22] (appearing under "1-bit", as it maintains a single bit per hash used) and the sketch of [14] ("FPS", after the names of the authors). The Figure indicates the massive saving in computation time our approach offers over the approach of [22], and shows that the running time of our approach and that of [14] is very similar.

We now examine the accuracy achieved by our approach, which depends on the sketch size. To analyze empirical accuracy, we isolated users who provided ratings for

[7] The Netflix data can easily be stored on disk. It is even possible to store it in memory on machine with a large RAM, by compressing it or using a sparse matrix representation.

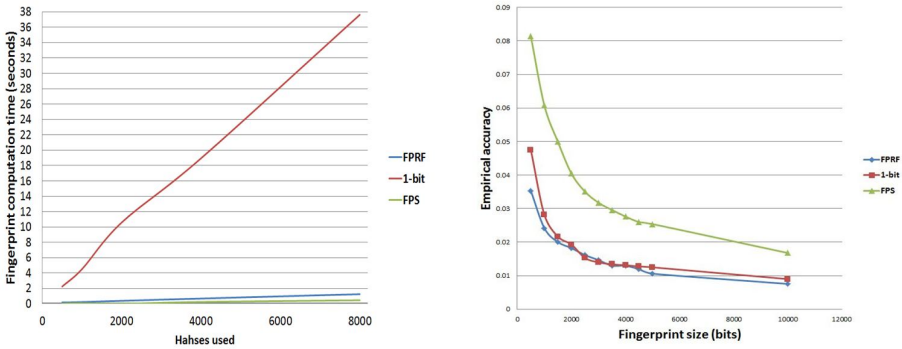

Fig. 2. Left: running time of fingerprint computation. Right: accuracy depending on size.

over 1,000 movies. There are over 10,000 such users in the dataset, and as these users have rated many movies, the Jaccard similarity between two such users is very fine-grained. We tested the fingerprint size required to achieve a target accuracy level for the Jaccard similarity. Consider a fingerprint size of k bits. Given two users, denote the true Jaccard similarity between their lists of rated movies as J. J can be easily computed using the entire dataset. Alternatively, we can use a fingerprint of size k, resulting in an estimate $\hat{J}$ that has a certain error. The error for a pair of users is $e = |J - \hat{J}|$. We can sample many such user pairs, and examine the average error obtained using a fingerprint of size k, which we call the *empirical inaccuracy*. We wish to minimize the error in our estimates, but to reduce the inaccuracy we must use larger fingerprints. As each user in our sample rated at least 1,000 movies, storing the full list of rated movies for a user takes 1,000 integers. The Netflix dataset only has 17,000 movies, so we require at least 15 bits to store the ID of each movie. Thus the full data for a user takes *at least* 15,000 bits. The space required for this data grows linearly with the numbers of movies a user has rated. Increasing the size of the universe of movies also increases the storage requirements, as more bits would be required to represent the ID of each movie.

Using our sketch, the required space does not depend on the number of ratings per user, or on the number of movies in the system, but rather on the target accuracy for the similarity estimate. Earlier sketches [5,3] eliminated the dependency on the number of ratings, but not on the number of movies in the system. Also, our fingerprints are faster to compute.

We tested how the average accuracy of our Jaccard similarity estimates changes as we chage the fingerprint size. We have tried fingerprints of different sizes, ranging from 500 bits to 10,000 bits. For each such size we sampled many pairs of users, and computed the average inaccuracy of the Jaccard similarity estimates. The results are given in Figure 2, for both our approach and for the sketch of [14] ("FPS"), as well as the "1-bit" sketch of [22]. Lower numbers indicate better empirical accuracy. Figure 2 shows that our sketch achieves a very high accuracy in estimating Jaccard similarity, even for small fingerprints. Even for a fingerprint size of 2500 bits per user, the Jaccard similarity can be estimated with an error smaller than 1.5%. Thus using fingerprints reduces the required storage to roughly 10% of that of the full dataset, without sacrifising much

accuracy in estimating user similarity. The figure also indicates that for any sketch size, the accuracy achieved by our approach is superior to that of the FPS sketch [14]. This is predictable since the theoretical accuracy guarantee for our approach is better than that for the sketch of [14]. The figure shows no significant difference in accuracy between our sketch and the 1-bit sketch [22].

Figure 2 shows that on the Netflix dataset, our sketch has the good properties of the mutually exclusive sketches of [22,14], and outperforms each of these state of the art methods in either running time or accuracy. The Netflix dataset is a small dataset, and the saving in space is much greater for larger datasets. A recommender system for web pages is likely to have several orders of magnitude more users and information items. While the storage requirements for such a massive recommender system grow by several orders of magnitude when storing the full data, the required space remains almost the same using our sketch. Previous approaches [5,3] compute the sketch in time *quadratic* in the required accuracy. Using our approach, computing the sketch only requires time *logarithmic* in the accuracy, which makes it tractable even when the required accuracy is very high.

4 Conclusions

We presented a fast method for sketching massive datasets, based on pseudo-random hashes. Though we focused on collaborative filtering and examined the Jaccard similarity in detail, the same technique can be used for any fingerprint based on minimal elements under several hashes. Our approach is thus a general technique for exponentially speeding up computation of various fingerprints, while maintaining a single bit per hash. We showed that even for these small fingerprints which can be quickly computed, the required number of hashes is asymptotically similar to previously known methods, and is logarithmic in the required confidence and polynomial in the required accuracy. Our empirical analysis shows that for the Netflix dataset the required storage is even smaller than the theoretical bounds.

Several questions remain open. Can we speed up the sketch computation further? Can similar methods be used that are not based on minimal elements under hashes?

References

1. Aggarwal, C.C.: Data streams: models and algorithms. Springer-Verlag New York Inc. (2007)
2. Alon, N., Matias, Y., Szegedy, M.: The Space Complexity of Approximating the Frequency Moments. J. Computer and System Sciences 58(1), 137–147 (1999)
3. Bachrach, Y., Herbrich, R.: Fingerprinting Ratings for Collaborative Filtering — Theoretical and Empirical Analysis. In: Chavez, E., Lonardi, S. (eds.) SPIRE 2010. LNCS, vol. 6393, pp. 25–36. Springer, Heidelberg (2010)
4. 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 2009. LNCS, vol. 5721, pp. 344–352. Springer, Heidelberg (2009)
5. Bachrach, Y., Porat, E., Rosenschein, J.S.: Sketching techniques for collaborative filtering. In: IJCAI, Pasadena, California (July 2009)

6. Bennett, J., Lanning, S.: The netflix prize. In: KDD Cup and Workshop (2007)
7. Broder, A.Z.: On the resemblance and containment of documents. Sequences (1998)
8. Broder, A.Z., Charikar, M., Frieze, A.M., Mitzenmacher, M.: Min-wise independent permutations. Journal of Computer and System Sciences 60(3), 630–659 (2000)
9. Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. Journal of Algorithms 55(1), 58–75 (2005)
10. Cormode, G., Muthukrishnan, S., Rozenbaum, I.: Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In: VLDB (2005)
11. Das, A.S., Datar, M., Garg, A., Rajaram, S.: Google news personalization: scalable online collaborative filtering. In: WWW. ACM (2007)
12. Dasgupta, A., Kumar, R., Sarlos, T.: Fast locality-sensitive hashing. In: SIGKDD (2011)
13. Datar, M., Muthukrishnan, S.: Estimating rarity and similarity over data stream windows. In: Möhring, R., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 323–335. Springer, Heidelberg (2002)
14. Feigenblat, G., Shiftan, A., Porat, E.: Exponential time improvement for min-wise based algorithms. In: SODA (2011)
15. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association 58(301), 13–30 (1963)
16. Indyk, P.: A Small Approximately Min-Wise Independent Family of Hash Functions. Journal of Algorithms 38(1), 84–90 (2001)
17. Indyk, P.: Stable distributions, pseudorandom generators, embeddings, and data stream computation. Journal of the ACM (JACM) 53(3), 323 (2006)
18. Kane, D.M., Nelson, J., Porat, E., Woodruff, D.P.: Fast moment estimation in data streams in optimal space. In: STOC (2011)
19. Kane, D.M., Nelson, J., Woodruff, D.P.: An optimal algorithm for the distinct elements problem. In: PODS, pp. 41–52. ACM (2010)
20. Karp, R.M., Shenker, S., Papadimitriou, C.H.: A simple algorithm for finding frequent elements in streams and bags. ACM Transactions on Database Systems (TODS) 28(1), 51–55 (2003)
21. Kirsch, A., Mitzenmacher, M.: Less hashing, same performance: a better Bloom filter. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 456–467. Springer, Heidelberg (2006)
22. Li, P., Koenig, C.: b-Bit minwise hashing. In: WWW (2010)
23. Mulmuley, K.: Randomized geometric algorithms and pseudorandom generators. Algorithmica (1996)
24. Pătraşcu, M., Thorup, M.: On the k-Independence Required by Linear Probing and Minwise Independence. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 715–726. Springer, Heidelberg (2010)
25. Pavan, A., Tirthapura, S.: Range-efficient counting of distinct elements in a massive data stream. SIAM Journal on Computing 37(2), 359–379 (2008)
26. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., Riedl, J.: Grouplens: an open architecture for collaborative filtering of netnews. In: Computer Supported Cooperative Work (1994)
27. Sarwar, B., Karypis, G., Konstan, J., Reidl, J.: Item-based collaborative filtering recommendation algorithms. In: WWW (2001)
28. Su, X., Khoshgoftaar, T.M.: A survey of collaborative filtering techniques. Advances in Artificial Intelligence 2009, 4 (2009)