Similarity Search for Dynamic Data Streams

Marc Bury, Chris Schwiegelshohn, and Mara Sorella

Abstract—Nearest neighbor searching systems are an integral part of many online applications, including but not limited to pattern recognition, plagiarism detection, and recommender systems. With increasingly larger data sets, scalability has become an important issue. Many of the most space and running time efficient algorithms are based on locality-sensitive hashing. Here, we view the data set as an n by $|U|$ matrix where each row corresponds to one of n users and the columns correspond to items drawn from a universe U. The de-facto standard approach to quickly answer nearest neighbor queries on such a data set is usually a form of min-hashing. Not only is min-hashing very fast, but it is also space efficient and can be implemented in many computational models aimed at dealing with large data sets such as MapReduce and streaming. However, a significant drawback is that minhashing and related methods are only able to handle insertions to user profiles and tend to perform poorly when items may be removed. We initiate the study of scalable locality-sensitive hashing (LSH) for fully dynamic data-streams. Specifically, using the Jaccard index as similarity measure, we design (1) a collaborative filtering mechanism maintainable in dynamic data streams and (2) a sketching algorithm for similarity estimation. Our algorithms have little overhead in terms of running time compared to previous LSH approaches for the insertion only case, and drastically outperform previous algorithms in case of deletions.

Index Terms—Dynamic streaming, locality-sensitive hashing, nearest neighbor searching

1 Introduction

Finding the most interesting pairs of points, i.e., typically those having small distance or, conversely, of high similarity, known as the *nearest-neighbor search* problem, is a task of primary importance, that has many applications such as plagiarism detection [2], clustering [3] and association rule mining [4]. The aim is to maintain a data structure such that we can efficiently report all neighbors within a certain distance from a candidate point. *Collaborative filtering* [5] is an approach to produce such an item set by basing the recommendation on the most similar users in the data set and suggesting items not contained in the intersection. To apply such an approach, one typically requires two things: (1) a measure of similarity (or dissimilarity) between users and (2) scalable algorithms for evaluating these similarities. In these contexts scalability can mean fast running times, but can also require strict space constraints.

Though it is by no means the only method employed in this line of research, locality-sensitive hashing (LSH) satisfies both requirements [4], [6]. For a given similarity measure, the algorithm maintains a small number of hash-values, or fingerprints that represent user behavior in a succinct way. The name implies that the fingerprints have locality properties, i.e., similar users have a higher probability of sharing the same fingerprint whereas dissimilar users have a small probability of agreeing on a fingerprint. The fingerprints themselves allow the recommendation system to quickly filter out user pairs with low similarity, leading to running times that are almost linear in input and output size.

A crucial property of LSH-families is that they are data-oblivious, that is the properties of the hash family depend only on the similarity measure but not on the data. Therefore, LSH-based filtering can be easily facilitated in online and streaming models of computation, where user attributes are added one by one in an arbitrary order. The fingerprint computation may fail, however, if certain attributes get deleted. Attribute deletion occurs frequently, for instance, if the data set evolves over time. Amazon allows users to unmark certain bought items for recommendations, Twitter users have an unfollow option, Last.fm users may delete songs or artists from the library. A naive way to incorporate deletions within the context of LSH is to recompute any affected fingerprint, which requires scanning the entire user profile and is clearly infeasible.

1.1 Contributions

We initiate the study of locality-sensitive nearest neighbors search in the dynamic data-stream model. Our input consists of sequence of triples (i, j, k), where $i \in [n]$ is the user identifier, $j \in |U|$ is the item identifier and $k \in \{-1, 1\}$ signifying insertion or deletion. Instead of maintaining an $n \times |U|$ user/attribute matrix, we keep a sketch of $\mathrm{polylog}(n \cdot |U|)$ bits per user.

In a first step, we show that the Jaccard distance $1 - \frac{|A \cap B|}{|A \cup B|}$ can be $(1 \pm \varepsilon)$-approximated in dynamic streams. Moreover, the compression used in this approximation is a black-box application of ℓ_0 sketches, which allows for extremely efficient algorithms from theory and practice. This also enables us to efficiently compress the n by n distance matrix using

- *M. Bury is with TU Dortmund, Dortmund 44227, Germany. E-mail: marc.bury@tu-dortmund.de.*
- *C. Schwiegelshohn and M. Sorella are with the Sapienza University of Rome, Rome 00185, Italy. E-mail: {schwiegelshohn, sorella}@diag.uniroma1.it.*

Manuscript received 13 Aug. 2018; revised 31 Mar. 2019; accepted 28 Apr. 2019. Date of publication 14 May 2019; date of current version 6 Oct. 2020. (Corresponding author: Chris Schwiegelshohn.)
Recommended for acceptance by K. Yi.
Digital Object Identifier no. 10.1109/TKDE.2019.2916858

n polylog $(n|U|)$ bits, similar in spirit to the compression by Indyk and Wagner [7] for euclidean spaces. This enables us to run any distance matrix-based algorithm in a dynamic semi-streaming setting.

Known lower bounds on space complexity of set intersection prevent us from achieving a compression with multiplicative approximation ratio for Jaccard similarity, see for instance [8]. From the multiplicative approximation for Jaccard distance we nevertheless get an ε-additive approximation to Jaccard similarity, which may be sufficient if the interesting similarities are assumed to exceed a given threshold. The same reduction further extends to a wide class of similarity functions on sets such as *rational set similarities* and *root similarities*, see Gower and Legendre [?]. However, even with this assumption, such a compression falls short of the efficiency we are aiming for, as it is not clear whether the relevant similarities can be found more quickly than by evaluating all similarities.

Our main contribution lies now in developing a compression scheme that simultaneously supports locality-sensitive hashing while satisfying a weaker form of approximation ratio. The construction is inspired by bit-hashing techniques used both by ℓ_0 sketches and min-hashing. In addition, our approach can be extended to similarities admitting LSHs other than min-hashing, such as Hamming, Anderberg, and Rogers-Tanimoto similarities. This approach, despite having provable bounds that are weaker than ℓ_0 sketches from an approximation point of view, is extremely simple to implement. Our implementation further shows that our algorithms have none to little overhead in terms of running time compared to previous LSH approaches for the insertion only case, and drastically outperform previous algorithms in case of deletions.

2 Preliminaries

We have n users and a universe set U of items. A *dynamic stream* consists of sequence of triples (i, j, k), where $i \in [n]$ is the user identifier, $j \in |U|$ is the item identifier and $k \in \{-1, 1\}$ signifying deletion or insertion, respectively. A user profile is a subset of U.

The symmetric difference of two sets $A, B \subseteq U$ is $A \triangle B = (A \setminus B) \cup (B \setminus A)$. The complement is denoted by $\overline{A} = U \setminus A$. Given $x, y \geq 0$ and $0 \leq z \leq z'$, the *rational set similarity* $S_{x,y,z,z'}$ between two item sets A and B is

$$S_{x,y,z,z'}(A,B) = \frac{x \cdot |A \cap B| + y \cdot |\overline{A \cup B}| + z \cdot |A \triangle B|}{x \cdot |A \cap B| + y \cdot |\overline{A \cup B}| + z' \cdot |A \triangle B|}$$

if it is defined and 1 otherwise. The distance function induced by a similarity $S_{x,y,z,z'}$ is defined as $D_{x,y,z,z'}(A,B) := 1 - S_{x,y,z,z'}(A,B)$. If $D_{x,y,z,z'}$ is a metric, we call $S_{x,y,z,z'}$ a metric rational set similarity [9]. The arguably most well-known rational set similarity is the Jaccard index $S(A,B) = S_{1,0,0,1}(A,B) = \frac{|A \cap B|}{|A \cup B|}$. A *root similarity* is defined as $S^{\alpha}_{x,y,z,z'} := 1 - (1 - S_{x,y,z,z'})^{\alpha}$ for any $0 < \alpha \leq 1$. We denote numerator and denominator of a rational set similarity by $Num(A,B)$ and $Den(A,B)$, respectively. For some arbitrary but fixed order of the elements, we represent A via its characteristic vector $a \in \{0,1\}^{|U|}$ with $a_i = 1$ iff $i \in A$. The ℓ_p-norm of a vector $a \in \mathbb{R}^d$ is defined as $\ell_p(a) = \sqrt[p]{\sum_{i=1}^{d} |a_i|^p}$. Taking the limit of p to 0, $\ell_0(x)$ is exactly the number of non-zero entries, i.e., $\ell_0(a) = |\{i \mid a_i \neq 0\}|$.

An LSH for a similarity measure $S : U \times U \to [0,1]$ is a set of hash functions H on U with an associated probability distribution such that

$$\mathbf{Pr}[h(A) = h(B)] = S(A,B)$$

for h drawn from H and any two item sets $A, B \subseteq U$. We will state our results in a slightly different manner. A (r_1, r_2, p_1, p_2)-sensitive hashing scheme for a similarity measure aims to find a distribution over a family of hash functions H such that for h drawn from H and two item sets $A, B \subseteq U$ we have

$$\mathbf{Pr}[h(A) = h(B)] \geq p_1 \text{ if } S(A,B) \geq r_1$$

and

$$\mathbf{Pr}[h(A) = h(B)] \leq p_2 \text{ if } S(A,B) \leq r_2.$$

The former definition due to Charikar [10] has a number of appealing properties and is a special case of the latter definition due to Indyk and Motwani [11]. Unfortunately, it is also a very strong condition and in fact not achievable for dynamic data streams. We emphasize that the general notions behind both definitions are essentially the same.

3 Related Work

3.1 Locality-Sensitive Hashing

Locality-sensitive hashing describes an algorithmic framework for fast (approximate) nearest neighbor search in metric spaces. In the seminal paper by Indyk and Motwani [11], it was proposed as a way of coping with the *curse of dimensionality* for proximity problems in high-dimensional euclidean spaces. The later, simpler definition by Charikar [10] was used even earlier in the context of min-hashing for the Jaccard index by Broder et al. [12], [13], [14]. Roughly speaking, min-hashing computes a fingerprint of a binary vector by permuting the entries and storing the first non-zero entry. For two item sets A and B, the probability that the fingerprint is identical is equal to the Jaccard similarity of A and B. When looking for item sets similar to some set A, one can arrange multiple fingerprints to filter out sets of small similarity while retaining sets of high similarity (see Cohen et al. [4], and Leskovec et al. [15] for details). We note that while this paper is focused mainly on min-hashing, locality-sensitive hashing has been applied to many different metrics, see Andoni and Indyk [16] for an overview.

Instead of using multiple independent hash functions to generate k fingerprints, Cohen and Kaplan suggested using the k smallest entries after a single evaluation [17], [18] which is known as bottom k-sampling, see also Hellerstein et al. [19]. Min-hashing itself is still an active area of research. Broder et al. [14] showed that an ideal min-hash family is infeasible to store, which initiated the search for more feasible alternatives. Indyk considered families of approximate min-wise independence [20], i.e., the probability of an item becoming the minimum is not uniform, but close to uniform, see also Feigenblat et al. [21].

Instead of basing requirements on the hash-function, other papers focus on what guarantees are achievable by simpler, easily stored and evaluated hash-functions with limited independence. Of particular interest are 2-wise independent hash functions. Dietzfelbinger [22] showed that, given two random numbers a and b, the hash of x may be computed via $(ax+ b) \gg k$, where k is a power of 2 and $\gg$ denotes a bit shift operation. This construction is to the best of our knowledge the fastest available and there exists theoretical evidence which supports that it may be, in many cases, good enough. Chung et al. [23] showed that if the entropy of the input is large enough, the bias incurred by 2-wise independent hash functions becomes negligible. Thorup [24] further showed that 2-wise independent hashing may be used for bottom k sampling with a relative error of $\frac{1}{\sqrt{fk}}$, where f is the Jaccard similarity between two items. Better bounds and/or running times are possible using more involved hash functions such as tabulation hashing [25], [26], linear probing [27], [28], one-permutation hashing [29], [30], [31], and feature hashing [32], [33].

3.2 Profile Sketching

Using the index of an item as a fingerprint is immediate and requires $\log_2|U|$ space. For two item sets A, B, we then require roughly $\frac{\log_2|U|}{\varepsilon^2 \cdot S(A,B)}$ bits of space to get an $(1 \pm \varepsilon)$-approximate estimate of the similarity $S(A, B)$. It turns out that this is not optimal, Bachrach and Porat [34], [35] and Li and König [36] proposed several improvements and constant size fingerprints are now known to exist. Complementing these upper bounds are lower bounds by Pagh [8] who showed that that this is essentially optimal for summarizing Jaccard similarity.

We note that one of the algorithms proposed by Bachrach and Porat in [34] use an ℓ_2 estimation algorithm as a black box to achieve fingerprint size of size $\frac{(1-S(A,B))^2}{\varepsilon^2}\log_2|U|$ bits. It is well known that the ℓ_2 norm of a vector can be maintained in dynamic data streams. However, their algorithm only seems to work for if the similarity is sufficiently large, i.e., $S(A, B) \geq 0.5$ and it does not seem to support locality-sensitive hashing.

4 Similarity Sketching in Dynamic Streams

In this section, we aim to show that the distance function of any rational set similarity with an LSH can be $(1 \pm \epsilon)$-approximated in dynamic streams. First, we recall the following theorem relating LSH-ability and properties of the induced dissimilarity:

Theorem 1. *Let $x, y, z, z' > 0$. Then the following three statements are equivalent.*

1) *$S_{x,y,z,z'}$ has an LSH.*
2) *$1 - S_{x,y,z,z'}$ is a metric.*
3) *$z' \geq \max(x, y, z)$.*

(1)⇒(2) was shown by Charikar [10], (2)⇒(1) was shown by Chierichetti and Kumar [37] and (2)⇔(3) was proven by Janssens [9]. We also recall the state of the art of ℓ_0 sketching in dynamic streams.

Theorem 2 (Th. 10 of Kane, Nelson, and Woodruff [38]). *There is a dynamic streaming algorithm for $(1 \pm \varepsilon)$-approximating $\ell_0(x)$ of a $|U|$-dimensional vector x using space $O(\frac{1}{\varepsilon^2}\log_2|U|)$,[1] with probability $2/3$, and with $O(1)$ update and query time.*

With this characterization, we prove the following.

Theorem 3. *Given a constant $0 < \varepsilon \leq 0.5$, two item sets $A, B \subseteq U$ and some rational set similarity $S_{x,y,z,z'}$ with metric distance function $1 - S_{x,y,z,z'}$, there exists a dynamic streaming algorithm that maintains a $(1 \pm \varepsilon)$ approximation to $1 - S_{x,y,z,z'}(A, B)$ with constant probability. The algorithm uses $O(\frac{1}{\varepsilon^2}\log_2|U|)$ space and has $O(1)$ update and query time.*

Proof. We start with the observation that $|A \triangle B| = \ell_0(a - b)$ and $|A \cup B| = \ell_0(a + b)$, where a and b are the characteristic vectors of A and B, respectively. Since $Den(A, B) - Num(A, B) = (z' - z) \cdot |A \triangle B|$ is always non-negative due to $z' \geq z$, we only have to prove that $Den(A, B)$ is always a non-negative linear combination of terms that we can approximate via sketches. First, consider the case $x \geq y$. Reformulating $Den(A, B)$, we have

$$Den(A, B) = y \cdot |U| + (x - y) \cdot |A \cup B| + (z' - x) \cdot |A \triangle B|.$$

Then both numerator and denominator of $1 - S_{x,y,z,z'}$ can be written as a non-negative linear combination of n, $|A \triangle B|$ and $|A \cup B|$. Given a $(1 \pm \varepsilon)$ of these terms, we have an upper bound of $\frac{1+\varepsilon}{1-\varepsilon} \leq (1 + \varepsilon) \cdot (1 + 2\varepsilon) \leq (1 + 5\varepsilon)$ and a lower bound of $\frac{1-\varepsilon}{1+\varepsilon} \geq (1 - \varepsilon)^2 \geq (1 - 2\varepsilon)$ for any $\varepsilon \leq 0.5$.

Now consider the case $x < y$. We first observe

$$S_{x,y,z,z'}(A, B) = S_{y,x,z,z'}(\overline{A}, \overline{B}).$$

Therefore

$$Den(A, B) = (y - x) \cdot |\overline{A} \cup \overline{B}| + x \cdot |U| + (z' - y) \cdot |\overline{A} \triangle \overline{B}|.$$

Again, we can write the denominator as a non-negative linear combination of $|\overline{A} \triangle \overline{B}|$, n and $|\overline{A} \cup \overline{B}|$. Dynamic updates can maintain an approximation of $|\overline{A} \triangle \overline{B}|$ and $|\overline{A} \cup \overline{B}|$, leading to upper and lower bounds on the approximation ratio analogous to those from case $x \geq y$.

By plugging in the ℓ_0 sketch of Theorem 2 and rescaling ε by a factor of 5, the theorem follows. □

Using a similar approach, we can approximate the distance of root similarity functions admitting a locality hashing scheme. We first repeat the following characterization.

Theorem 4 (Theorem 4.8 and 4.9 of [37]). *The root similarity $S^{\alpha}_{x,y,z,z'}$ is LSH-able if and only if $z' \geq \frac{\alpha+1}{2}\max(x, y)$ and $z' \geq z$.*

Theorem 5. *Given a constant $0 < \varepsilon \leq 0.5$, two item sets $A, B \subseteq U$ and some LSH-able root similarity $S^{\alpha}_{x,y,z,z'}$, there exists a dynamic streaming algorithm that maintains a $(1 \pm \varepsilon)$ approximation to $1 - S^{\alpha}_{x,y,z,z'}(A, B)$ with constant probability. The algorithm uses $O(\frac{1}{\varepsilon^2}\log_2|U|)$ space and each update and query requires $O(1)$ time.*

1. The exact space bounds of the ℓ_0 sketch by Kane, Nelson, and Woodruff depends on the magnitude of the entries of the vector. The stated space bound is sufficient for our purposes as we are processing binary entries.

Proof. We consider the case $x \geq y$, the case $y \geq x$ can be treated analogously. Again we will show that we can $(1 \pm \varepsilon)$-approximate the denominator; the remaining arguments are identical to those of Theorem 3. Consider the following reformulation of the denominator

$$Den(A, B) = y \cdot n + (x - z') \cdot |A \cap B| + (z' - y) \cdot |A \cup B|.$$

We first note that we can obtain an estimate of $|A \cap B|$ in a dynamic data stream with additive approximation factor $\varepsilon \cdot |A \cup B|$ by computing $|A| + |B| - \widehat{|A \cup B|}$, where $\widehat{|A \cup B|}$ is a $(1 \pm \varepsilon)$-approximation of $|A \cup B|$.

Due to Theorem 4, we have $x - z' \leq 2 \cdot z' - z' \leq z'$ and either $z' - y \geq \frac{z'}{2}$ or $y \geq \frac{z'}{2}$. Hence $\varepsilon \cdot (x - z') \leq \varepsilon \cdot z' \leq 2\varepsilon \cdot \max(z', (z' - y))$. Since further $|U| \geq |A \cup B|$, we then obtain a $(1 \pm 2\varepsilon)$-approximation to the denominator. Rescaling ε completes the proof. □

Remark 1. Theorems 3 and 5 are not a complete characterization of dissimilarities induced by similarities that can be $(1 \pm \varepsilon)$-approximated in dynamic streams. Consider, for instance, the Sørenson-Dice coefficient $S_{2,0,0,1} = \frac{2 \cdot |A \cap B|}{|A| + |B|}$ with $1 - S_{2,0,0,1} = \frac{|A \triangle B|}{|A| + |B|}$. Neither is $1 - S_{2,0,0,1}$ a metric, nor do we have $z' \geq \frac{\alpha+1}{2} x$ for any $\alpha > 0$. However, both numerator and denominator can be approximated using ℓ_0 sketches.

The probability of success can be further amplified to $1 - \delta$ in the standard way by taking the median estimate of $O(\log_2(1/\delta))$ independent repetitions of the algorithm. For n item sets, and setting $\delta = 1/n^2$, we then get the following corollary.

Corollary 1. *Let S be a rational set similarity with metric distance function $1 - S$. Given a dynamic data stream consisting of updates of the form $(i, j, v) \in [n] \times [|U|] \times \{-1, +1\}$ meaning that $a_j^{(i)} = a_j^{(i)} + v$ where $a^{(i)} \in \{0, 1\}^{|U|}$ with $i \in \{1, \ldots, n\}$, there is a streaming algorithm that can compute with constant probability for all pairs (i, i')*

- *a $(1 \pm \varepsilon)$ multiplicative approximation of $1 - S(a^i, a^{i'})$ and*
- *an ϵ-additive approximation of $S(a^i, a^{i'})$.*

The algorithm uses $O(n \log_2 n \cdot \varepsilon^{-2} \cdot \log_2 |U|)$ space and each update and query needs $O(\log_2 n)$ time.

We note that despite the characterization of LSH-able rational set similarities of Theorem 1, the existence of the approximations of Corollary 1 hints at, but does not directly imply the existence of a locality-sensitive hashing scheme or even an approximate locality-sensitive hashing scheme on the sketched data matrix in dynamic streams. Our second and main contribution now lies in the design of a simple LSH scheme maintainable in dynamic data streams, albeit with weaker approximation ratios. The scheme is space efficient, easy to implement and to the best of our knowledge the first of its kind able to process deletions.

Remark 2. Corollary 1 also implies that any algorithm based on the pairwise distances of a rational set similarity admits a dynamic streaming algorithm using $n \cdot \text{ polylog } (n|U|)$ bits of space. Notable examples include hierarchical clustering algorithms such as single or complete linkage, distance matrix methods used in phylogeny, and visualization methods such as heatmaps. Though the main focus in the experimental section (Section 6) will be an evaluation of the dynamic hashing performance, we also briefly explore clustering and visualization methods based on the sketched distance matrix.

5 An LSH Algorithm for Dynamic Data Streams

In the following, we will present a simple dynamic streaming algorithm that supports Indyk and Motwani-type sensitivity. Recall that we want to find pairs of users with similarity greater than a parameter r_1, while we do not want to report pairs with similarity less than r_2. The precise statement is given via the following theorem.

Theorem 6. *Let $0 < \varepsilon, \delta, r_1, r_2 < 1$ be parameters. Given a dynamic data stream with n users and $|U|$ attributes, there exists an algorithm that maintains a $(r_1, r_2, (1 - \varepsilon)r_1, 6r_2/(\delta(1 - \varepsilon/5\sqrt{2r_1}))$-sensitive LSH for Jaccard similarity with probability $1 - \delta$. For each user, $O(\frac{1}{\varepsilon^4 \delta^5 \cdot r_1^2} \log_2^2 |U|)$ bits of space are sufficient. The update time is $O(1)$.*

The proof of this theorem consists of two parts. First, we give a probabilistic lemma from which we derive the sensitivity parameters. Second, we describe how the sampling procedure can be implemented in a streaming setting.

5.1 Sensitivity Bounds

While a black box reduction from any ℓ_0 sketch seems unlikely, we note that most ℓ_0 algorithms are based on bit-sampling techniques similar to those found in min-hashing. Our own algorithm is similarly based on sampling a sufficient number of bits or item indexes from each item set. Given a suitably filtered set of candidates, these indexes are then sufficient to infer the similarity. Let $U_k \subseteq U$ be a random set of elements where each element is included with probability 2^{-k}. Further, for any item set A, let $A_k = A \cap U_k$. Note that in $S_{x,y,z,z'}(A_k, B_k)$ the value of $|U|$ is replaced by $|U_k|$. At the heart of the algorithm now lies the following technical lemma.

Lemma 1. *Let $0 < \varepsilon, \delta, r < 1$ be constants and $S_{x,y,z,z'}$ be a rational set similarity with metric distance function. Let A and B be two item sets. Assume we sample every item uniformly at random with probability 2^{-k}, where $k \leq \log_2 \left(\frac{\varepsilon^2 \cdot \delta \cdot r \cdot Den_{x,y,z,z'}(A,B)}{100 \cdot z'} \right)$.*

Then with probability at least $1 - \delta$ the following two statements hold.

1) *If $S_{x,y,z,z'}(A, B) \geq r$ we have $(1 - \varepsilon) S_{x,y,z,z'}(A, B) \leq S(A_k, B_k) \leq (1 + \varepsilon) S_{x,y,z,z'}(A, B)$.*
2) *$S_{x,y,z,z'}(A_k, B_k) \leq \frac{2 \cdot S_{x,y,z,z'}(A,B)}{\delta(1 - (\varepsilon/5) \cdot \sqrt{2r})}$.*

We note that any metric distance function induced by a rational set similarity satisfies $z' \geq \max(x, y, z)$, see Theorem 1 in Section 4.

Proof. Let $Den_k = Den(A_k, B_k)$, $Num_k = Num(A_k, B_k)$, and $X_i = 1$ iff $i \in U_k$. If $S_{x,y,z,z}(A, B) \geq r$ then $Num(A, B) \geq r \cdot Den(A, B)$. Thus, we have $\mathbb{E}[Num_k] = Num(A, B)/2^k \geq r \cdot Den(A, B)/2^k$ and $\mathbb{E}[Den_k] = Den(A, B)/2^k$. Further, we

have $\mathbf{Var}[X_i] = 2^{-k} \cdot (1 - 2^{-k}) \leq 2^{-k}$. for any X_i. We first give a variance bound on the denominator.

$$\begin{aligned}
&\mathbf{Var}[Den_k] \\
&= \mathbf{Var}[x \cdot |A_k \cap B_k| + y \cdot (|U_k| - |A_k \cup B_k|)| \\
&\quad + z' \cdot |A_k \triangle B_k|] \\
&= x^2 \sum_{i \in A \cap B} \mathbf{Var}[X_i] + y^2 \sum_{i \in \overline{A \cup B}} \mathbf{Var}[X_i] \\
&\quad + z'^2 \sum_{i \in A \triangle B} \mathbf{Var}[X_i] \\
&= \big((x^2 - y^2)|A \cap B| + y^2 \cdot |U| \\
&\quad + (z'^2 - y^2)|A \triangle B|\big)\mathbf{Var}[X_i] \\
&\leq \big((x^2 - y^2)|A \cap B| + y^2 \cdot |U| + (z'^2 - y^2)|A \triangle B|\big)/2^k \\
&\leq \frac{1}{2^k}\max\{x + y, z' + y, y\} \cdot ((x - y)|A \cap B| + \\
&\quad y \cdot d + (z' - y)|A \triangle B|) \\
&\leq 2z' \cdot \mathbf{E}[Den_k]
\end{aligned}$$

and analogously

$$\begin{aligned}
&\mathbf{Var}[Num_k] \\
&= \mathbf{Var}[x \cdot |A_k \cap B_k| + y \cdot (|U_k| - |A_k \cup B_k|)| \\
&\quad + z \cdot |A_k \triangle B_k|] \\
&\leq \frac{1}{2^k}\max\{x + y, z + y, y\} \cdot ((x - y)|A \cap B| + \\
&\quad y \cdot d + (z' - y)|A \triangle B|) \\
&\leq \max\{x + y, z + y, y\} \cdot \mathbf{E}[Num_k].
\end{aligned}$$

Using Chebyshev's inequality we have

$$\begin{aligned}
&\mathbb{P}[|Den_k - \mathbf{E}[Den_k]| \geq \varepsilon/5 \cdot \mathbf{E}[Den_k]] \\
&\leq \frac{50z'}{\varepsilon^2 \cdot \mathbf{E}[Den_k]} \leq \frac{50z' \cdot 2^k}{\varepsilon^2 \cdot Num(A, B)},
\end{aligned}$$

and

$$\begin{aligned}
&\mathbb{P}[|Num_k - \mathbf{E}[Num_k]| \geq \varepsilon/5 \cdot \mathbf{E}[Num_k]] \\
&\leq \frac{25\max\{x + y, z + y, y\}}{\varepsilon^2 \cdot \mathbf{E}[Num_k]} \leq \frac{50z' \cdot 2^k}{\varepsilon^2 \cdot Num(A, B)}.
\end{aligned}$$

If $k \leq \log_2\left(\frac{\varepsilon^2 \delta r Den(A,B)}{100z'}\right) \leq \log_2\left(\frac{\varepsilon^2 \delta Num(A,B)}{100z'}\right)$ then both $Den_k - \mathbf{E}[Den_k] \leq \frac{\varepsilon}{5}\mathbf{E}[Den_k]$ and $Num_k - \mathbf{E}[Num_k] \leq \frac{\varepsilon}{5}\mathbf{E}[Num_k]$ hold with probability at least $1 - \delta/2$. Then we can bound $S(A_k, B_k) = Num_k/Den_k$ from above by

$$\begin{aligned}
&\frac{Num(A, B)/2^k + \varepsilon Num(A, B)}{Den(A, B)/2^k - \varepsilon Den(A, B)/2^k)} \\
&= \frac{1 + \varepsilon/5}{1 - \varepsilon/5} \cdot S_{x,y,z,z'}(A, B) \\
&\leq (1 + \varepsilon) \cdot S_{x,y,z,z'}(A, B).
\end{aligned}$$

Analogously, we can bound $S_{x,y,z,z'}(A_k, B_k)$ from below by $\frac{1-\varepsilon/5}{1+\varepsilon/5} \cdot S_{x,y,z,z'}(A, B) \geq (1 - \varepsilon) \cdot S_{x,y,z,z'}(A, B)$ which concludes the proof of the first statement.

For the second statement, we note that the expectation of Num_k can be very small because we have no lower bound on the similarity. Hence, we cannot use Chebyshev's inequality for an upper bound on Num_k. But it is enough to bound the probability that Num_k is greater than or equal to $(2/\delta) \cdot \mathbf{E}[Num_k]$ by $\delta/2$ using Markov's inequality. With the same arguments as above, we have that the probability of $Den_k \leq (1 - \varepsilon') \cdot \mathbf{E}[Den_k]$ is bounded by $\frac{\varepsilon^2 r \delta}{25 \cdot \varepsilon'^2}$ which is equal to $\delta/2$ if $\varepsilon' = \varepsilon/5 \cdot \sqrt{2r}$. Putting everything together we have that

$$S_{x,y,z,z'}(A_k, B_k) \leq \frac{2}{\delta(1 - (\varepsilon/5) \cdot \sqrt{2r})} \cdot S_{x,y,z,z'}(A, B)$$

with probability at least $1 - \delta$. □

We note that for similarities with $y > x$, we can obtain the same bounds by sampling 0-entries instead of 1-entries. Since we are not aware of any similarities with this property used in practice, we limited our analysis to the arguably more intuitive case $x \geq y$.

Applying this lemma on a few better known similarities gives us the following corollary. We note that to detect candidate high similarity pairs for an item set A, $Den := |A \cup B| \geq |A|$ for Jaccard and $Den := |A \cup B| + |A \triangle B| \geq |A|$ for Anderberg. For Hamming and Rogers-Tanimoto similarities, $Den \geq |U|$. More examples of rational set similarities can be found in Naish, Lee, and Ramamohanarao [39].

Corollary 2. *Let $\alpha := \varepsilon^2 \delta \cdot r$. Then if we sample items with at least probability 2^{-k}, the similarity is preserved for any two item sets A and B as per Lemma 1. The following table reports suitable values of k for interesting rational set similarities.*

Similarity	*Parameters*	k
Jaccard	$S_{1,0,0,1}$	$\log(\alpha|A|/100)$
Hamming	$S_{1,1,0,1}$	$\log(\alpha|U|/100)$
Anderberg	$S_{1,0,0,2}$	$\log(\alpha|A|/200)$
Rogers-Tanimoto	$S_{1,1,0,2}$	$\log(\alpha|U|/200)$

5.2 Streaming Implementation

When applying Corollary 2 or more generally Lemma 1 to a dynamic streaming environment, we have to address a few problems. First, we may not know how to specify the number of items we are required to sample. For Hamming and Rogers-Tanimoto similarities, it is already possible to run a black box LSH algorithm (such as the one by Cohen et al. [4]) if the number of sampled items are chosen via Corollary 2. For Jaccard (and Anderberg), the sample sizes depend on the cardinality of A, which requires additional preprocessing steps.

5.3 Cardinality-Based Filtering

As a first filter, we limit the candidate solutions based on their respective supports. For each item, we maintain the cardinality, which can be done exactly in a dynamic stream via counting. If the sizes of two item sets A and B differ by a factor of at least r_1, i. e., $|A| \geq r_1 \cdot |B|$, then the distance between these two sets has to be

$$1 - S(A, B) = \frac{|A \triangle B|}{|A \cup B|} \geq \frac{|A| - |B|}{|A|} \geq 1 - 1/r_1.$$

We then discard any item set with cardinality not in the range of $[r_1 \cdot |A|, |A|]$. Like the algorithm by Cohen et al. [4], we can do this by sorting the rows or hashing.

5.4 Small Space Item Sampling

Since the cardinality of an item set may increase and decrease as the stream is processed, we have to maintain multiple samples U_k in parallel for various values of k. If a candidate k is larger than the threshold given by Corollary 2, we will sample only few items and still meet a small space requirement. If k is too small, $|U_k|$ might be too large to store. We circumvent this using a nested hashing approach we now describe in detail.

Sampling with 2-Universal Hash Functions We first note that U_k does not have to be a fully independent randomly chosen set of items. Instead, we only require that the events X_i are pairwise independent. The only parts of the analysis of Lemma 1 that could be affected are the bounds on the variances, which continue to hold for pairwise independence. This allows us to emulate the sampling procedure using universal hashing. Assume that M is a power of 2 and let $h : [|U|] \to [M]$ be a 2-wise independent universal hash function, i.e., $\mathbb{P}[h(a) = j] = \frac{1}{M}$, for all $j \in [M]$. We set $U_k = \{j \in [M] \mid \mathrm{lsb}(h(j)) = k\}$, where $\mathrm{lsb}(x)$ denotes the first non-zero index of x when x is written in binary and $\mathrm{lsb}(0) = \log_2 M$. Since the image of h is uniformly distributed on $[M]$, each bit of $h(j)$ is 1 with probability $1/2$, and hence we have $\mathbb{P}[\mathrm{lsb}(h(j)) = k] = 2^{-k}$. Moreover, for any two j, j' the events that $\mathrm{lsb}(h(j)) = k$ and $\mathrm{lsb}(h(j')) = k$ are independent. The value of M may be adjusted for finer (M large) or coarser (M small) sampling probabilities. In our implementation (see Section 6) as well as in the proof of Theorem 6, we set $M = |U|$. Following Dietzfelbinger [22], h requires $\log_2|U|$ bits of space.

Recovery and Compression via Perfect Hashing To avoid storing the entire domain of h in the case of large $|U_k|$, we pick, for each $k \in [0, \ldots, \log_2|U|]$, another 2-wise independent universal hash function $h_k : [|U|] \to [c^2]$, for some absolute constant c to be specified later. For some $j \in [|U|]$, we first check if $\mathrm{lsb}(h(j)) = k$. If this is true, we apply $h_k(j)$.

For the ith item set, we maintain a set $T^i_{k,\bullet}$ of buckets $T^i_{k,h_k(j)}$ for all $k \in \{0, \ldots \log_2|U|\}$ and $h_k(j) \in \{0, \ldots, c^2 - 1\}$. Each such bucket $T^i_{k,h_k(j)}$ contains the sum of the entries hashed to it. This allows us to maintain the contents of $T^i_{k,h_k(j)}$ under dynamic updates. Note that to support similarity estimation for sets that might have a low cardinality at query time, we must also maintain a bucket set $T^i_{0,\bullet}$ associated to a hash function h_0, that will receive all items seen so far for a given set i, i. e., each of them will be hashed to the bucket $T^i_{0,h_0(j)}$ with probability $2^0 = 1$ (see line 7 in Algorithm 1).

For the interesting values of k, i. e., $k \in \Theta(\log_2|A|)$, the number of indexes sampled by h will not exceed some constant c. This means that the sampled indexes will be perfectly hashed by h_k, i. e., the sum contained in $T^i_{k,h_k(j)}$ consists of exactly one item index. If k is too small (i.e., we sampled too many indexes), h_k has the useful effect of compressing the used space, as c^2 counters require at most $O(c^2 \log_2|U|)$ bits of space.

We can then generate the fingerprint matrix, for instance, by performing a min-hash on the buckets $B^i_{k,\bullet}$ and storing the index of the first non zero bucket. For a pseudocode of this approach, see Algorithm 1. Algorithm 2 describes an example candidate generation as per Cohen et al. [4].

Algorithm 1. Dynamic Stream Update (Jaccard)

Input: Parameter $c \in \mathbb{N}$
Output: $T^{(i)}_{k,l}$ with $i \in [n], k \in [0, \ldots, \log_2 m], l \in [c^2]$
Initialization:
$s_i = 0$ for all $i \in [n]$
$T^{(i)}_{k,l} = 0$ for all $i \in [n], k \in [0, \ldots, \log_2|U|], l \in [c^2]$.
$h : [|U|] \to [M]$ a 2-universal hash function.
$h_1 : [M] \to [c^2]$ another 2-universal hash function.
1: **On update** (i, j, v):
2: $k = \mathrm{lsb}(h(j))$
3: $T^{(i)}_{k,h_1(j)} = T^{(i)}_{k,h_1(j)} + v$
4: $T^{(i)}_{0,h_1(j)} = T^{(i)}_{0,h_1(j)} + v$
5: $s_i = s_i + v$

Algorithm 2. Filter Candidates (Jaccard)

Input: Thresholds $0 < r_1, \alpha < 1$, $B^{(i)}_{k,l}$ from Alg.1 with $k \in \{0, 1, 2, \ldots, \log_2|U|\}$
Output: Set of candidate pairs
Initialization:
$I = \{0, \log_2(1/r_1), 2\log_2(1/r_1), \ldots, \log_2|U|\}$
H_i: empty list for $i \in I$.
1: **for** $i \in [n]$ **do**
2: $s = \ell_0(x^{(i)})$
3: **for** $k \in [\log_2(r_1 \cdot \alpha \cdot s), \log_2(\alpha \cdot s)] \cap I$ **do**
4: add $(i, MinHash(T^{(i)}_{k,\bullet}))$ to H_k
5: **end for**
6: **end for**
7: **return** $\{(i, i') \mid \exists k : (i, h), (i', h') \in H_k \text{ and } h = h'\}$

Proof of Theorem 6. Fix items sets A and B and let a, b be the corresponding characteristic vectors for the sets A and B, respectively. Without loss of generality, assume $|A| \geq |B|$. Set $\alpha = \frac{\varepsilon^2 \cdot \delta}{600}$. If $S(A, B) \geq r_1$ then $|A|/|B| \leq 1/r_1$, then $\log_2(\alpha \cdot |B|) \leq \log_2(\alpha \cdot |A|)$ and $\log_2(r_1 \cdot \alpha \cdot |B|) \leq \log_2(\alpha \cdot |A|)$. Both sets will then enter line 3 of Algorithm 2 for some common values of k, and must exist at least an H_k containing min-hashes from both sets as per line 4.

Let 2^k be the largest power of 2 such that $k \leq \log_2(\alpha \cdot r_1|A \cup B|) \leq \log_2(\alpha \cdot |A \cap B|)$. Let U_k be a subset of indexes as determined by line 4 of Algorithm 1 and define $A_k := U_k \cap A$ and $B_k := U_k \cap B$.

In expectation, $\mathbb{E}[|A_k \cup B_k|] = |A \cup B|/2^k$. By Markov's inequality, we have $|A_k \cup B_k| \leq \frac{3}{\delta} \cdot |A \cup B|/2^k \leq \frac{1800}{\varepsilon^2 \delta^2 \cdot r_1}$ with probability at least $1 - \delta/3$. By setting the number of buckets in the order of

$$c^2 = |A_k \cup B_k|^2 \in O\left(\frac{1}{\varepsilon^4 \delta^5 \cdot r_1^2}\right), \tag{1}$$

the elements of $A_k \cup B_k$ will be perfectly hashed by h_k with probability at least $1 - \delta/3$ (line 3 of Algorithm 1). Since deleting indexes where both vector entries are zero does not change the Jaccard similarity, the probability that the smallest index in the collection of buckets $T^{(p)}_{k,\bullet}$ is

equal to the smallest index in the collection of buckets $T_{k,\bullet}^{(q)}$ is equal to the similarity of A_k and B_k. Thus we have

$$\mathbb{P}[MinHash(T_{k,\bullet}^{(p)}) = MinHash(T_{k,\bullet}^{(q)})] = S(A_k, B_k).$$

If $S(A,B) \geq r_1$ we have by our choice of α and due to the first part of Lemma 1, $S(A_k, B_k) \geq (1-\varepsilon) \cdot S(A,B)$ with probability $1 - \frac{\delta}{3}$. If $S(A,B) \leq r_2 < r_1$, we have due to the second part of Lemma 1 $S(A_k, B_k) \leq \frac{6 \cdot S(A,B)}{\delta(1-(\varepsilon/5)\cdot\sqrt{2r_1})} \leq \frac{6r_2}{\delta(1-(\varepsilon/5)\cdot\sqrt{2r_1})}$ with probability $1 - \frac{\delta}{3}$.

Conditioning on all events gives us a $(r_1, r_2, (1-\varepsilon)r_1, 6r_2/(\delta(1-\varepsilon/5\sqrt{2r_1}))$-sensitive LSH with probability $1-\delta$.

To bound the space requirement, observe that for each of the n item sets we have $\log_2|U|$ collections $T_{k,\bullet}^{(p)}$ of $c^2 \in O\left(\frac{1}{\varepsilon^4\delta^5 \cdot r_1^2}\right)$ buckets due to Equation 1. Each bucket contains a sum that uses at most $\log_2|U|$ bits. The space required for each hash function is at most $\log_2|U|$ due to Dietzfelbinger [22].

For every item insertion or deletion, we execute lines 2-5 of Algorithm 1. Each of these operations are elementary arithmetic operations that run in constant time. □

The parameters in Theorem 6 can be chosen such that we are able to use Algorithm 1 and Algorithm 2 similar to the min-hashing technique in the non-dynamic scenario. This also means that we can use similar tricks to amplify the probability of selecting high similar items in Algorithm 2 and lower the probability in case of a small similarity as long as $(1-\varepsilon)r_1 > \frac{6r_2}{\delta(1-\varepsilon/5)\sqrt{r_1}}$, see also Leskovec et al. [15]. Let $\ell, m \in \mathbb{N}$. Then we repeat the hashing part of Algorithm 2 ℓ times and only add a pair to the output set iff all ℓ hash values are equal. This procedure is repeated m times and the final output set contains all pairs which appear at least once in an output set of the m repetitions. The probability that a pair with similarity s is in the output set is $1-(1-p^\ell)^m$ with $p \geq (1-\delta)(1-\varepsilon)s$ if $s > r_1$ and $p \leq 6s/(\delta(1-\varepsilon/5\sqrt{r_2})$ if otherwise.

6 Experimental Evaluation

In this section we evaluate the practical performance of the algorithm given in Section 5. Our aim is two-fold: First, we want to show that the running time of our algorithm is competitive with more conventional min-hashing algorithms. For our use-case, i. e., dynamic streams, we are not aware of any competitors in literature. Nevertheless, it is important to demonstrate the algorithm's viability, as in many cases a system might not even support a dynamic streaming environment: we show a performance comparison in Section 6.1. To cover all ranges of user profiles, we use a synthetic benchmark described below.

Our second goal is to be able to measure the quality of the algorithm's output. We deem our filtering mechanism to be successful if it finds most of the user pairs with high similarity, while performing a good level of filtering, returning as candidates few user pairs with low similarity. Furthermore, Theorem 6 guarantees us a reasonable approximation to the similarity of each pair, though it is unclear whether this still holds for all pairs simultaneously, especially for small bucket sizes. We are satisfied if our approximate computation based on sketches does not lead to high deviation with respect to exact similarities. As a typical candidate from practice, we consider profiles of users containing recently preferred artists from Last.FM.

Implementation Details. We implemented Algorithms 1, 2, as well as other hash routines in C++ and compiled the code with GCC version 4.8.4 and optimization level 3. Compared to the description of Algorithm 2, which has 5 parameters (error ε, failure probability δ, lower bound for desirable similarities r_1, upper bound for undesirable similarities r_2, and granularity of the sampling given by M), our implementation has only two parameters: (1) the inverse sampling rate α and (2) the number of buckets c^2. Recall that a higher inverse sampling rate α means selecting higher values of k in Algorithm 2, line 5, where an increasing k is associated to a decreasing sampling probability 2^{-k} of a bucket $T_{k,h_k(i)}$.

The choice of c^2 influences the possible combinations of ε, δ, and r_1, see Equation 1 for an upper bound on c^2. The cardinality based filtering of Algorithm 2 is influenced by the choice of α.

As a rule of thumb, r_2 is roughly of the order $r_1^{1.5}$. For example, if we aim to retain all pairs of similarity at least $\frac{1}{4}$, we can filter out pairs with similarity less than $\frac{1}{8}$. Pairs with an intermediate similarity, i.e., a similarity within the interval $[\frac{1}{8}, \frac{1}{4}]$, may or may not be detected. We view this as a minor restriction as it is rarely important for these thresholds to be sharp.

Lastly, we implemented Dietzfelbinger's multiply-add-shift method to generate 2-wise independent hash functions, where a is a random non-negative odd integer, b a random non-negative integer, and for a given M the shift is set to $w - \log_2(M)$, where w is the word size (32 bits in our implementation). All hash functions used in the implementation of both Algorithm 1, that is the functions h, h_1 and the hash functions used for implementing the MinHash scheme, with amplification parameters ℓ (functions in one band) and m (number of bands) at line 6 of Algorithm 2), are 2-wise independent hash functions, and were generated independently, i. e., we did not reuse them for subsequent experiments. Otherwise the implementation follows that of Algorithms 1 and 2 with various choices of parameters.

All computations were performed on a 2.7 GHz Intel Core i7 machine with 8 MB shared L3 Cache and 16 GB main memory. Each run was repeated 10 times.

Synthetic Dataset To accurately measure the distortion on large datasets, for varying feature spaces, we used the synthetic benchmark by Cohen et al. [4]. Here we are given a large binary data-matrix consisting of 10,000 rows and either 10,000, 100,000 or 1,000,000 columns. The rows corresponded to item sets and the columns to items, i. e., we compared the similarities of rows. Since large binary data sets encountered in practical applications are sparse, the number of non-zero entries of each row was between 1 to 5 percent chosen uniformly at random. Further, for every 100th row, we added an additional row with higher Jaccard similarity in the range of $\{(0.35, 0.45), (0.45, 0.55), (0.55, 0.65), (0.65, 0.75), (0.75, 0.85), (0.85, 0.95)\}$.

To obtain such a pair, we copied the preceding row (which was again uniformly chosen at random) and uniformly at random flipped an appropriate number of bits, e. g., for 10,000 items, row sparsity of 5 percent, and similarity range

TABLE 1
Distribution of Exact Similarity Values for Pairs in Last.fm Dataset

Similarity	0.0	0.05	0.1	0.15
No. of pairs	60432710	37947485	12031795	1938117
Similarity	0.2	0.25	0.3	0.35
No. of pairs	164246	7855	266	13
Similarity	0.4	0.45	0.5	≥ 0.55
No. of pairs	10	1	3	0

(0.45, 0.55) we deleted an item contained in row i with probability $1/3$ and added a new item with probability $\frac{1}{19} \cdot \frac{1}{3} = \frac{1}{57}$. In the insertion-only case, the stream consists of the sequence of 1-entries of each row. We introduced deletions by randomly removing any non-zero entry immediately after insertion with probability $\frac{1}{10}$.

Last.FM Dataset. For an evaluation of our algorithm on real data we considered a dataset from [40] containing temporal data from the popular online (social) music recommendation system Last.fm. Users update their profiles in multiple ways: listening to their personal music collection with a music player supporting the Last.fm *Audioscrobbler* plugin, or by listening to the Last.fm Radio service, either with Last.fm official client application, or with the web player. Radio stations consist of uninterrupted audio streams of individual tracks based on the user's profile, its "musical neighbors" (i. e., similar users identified by the platform), or the user's "friends". All songs played are added to a log from which personalized top charts and musical recommendations are calculated, using a collaborative filtering algorithm. This automated track logging process is called *scrobbling*. Our dataset contains the full "scrobbled" listening history of a set of 44,154 users, covering a period of 5-years (May 2009-May 2014), containing 721M listening events and around 4.6M unique tracks, where each track is labeled with a score for a set of 700 music genres. To obtain a more granular feature space, we decided to map each track to the corresponding artist. To this end we queried the MusicBrainz DB[2] to obtain artist information for each of the unique tracks (total of $1.2M$ artists). We then processed the listening histories of each user i in chronological order to produce our event stream, emitting a triple `(i,j,+1)` after a user has listened to at least 5 songs by an artist j, and emitting a triple `(i,j,-1)` when no track from artist j is listened by i for a period of 6 months (expiration time). The threshold of 5 tracks is mainly intended to mitigate the "recommendation effect": being Last.fm a recommendation system, some portions of the listening histories might in fact be driven by recommendation sessions, where diverse artists are suggested by the system based on the user's interests (i. e., not explicitly chosen by him), and are likely to lead to cascades of deletions in the stream after the expiration time. Like most real world datasets that link users to bought/adopted items, this dataset is very sparse. For rows with only sparse support, a fast, space efficient nearest neighbor data structure typically does not improve over a naive approach that simply stores everything. We therefore only considered only users having at least 0.5 percent-dense profiles on average, obtaining a final set of $n = 15K$ users (sets), $|U| = 380K$ (items) and a stream length of $6.2M$ entries. Table 1 shows the distribution of exact similarity values for all pairs the Last.fm dataset.

2. https://musicbrainz.org/doc/MusicBrainz_Database

6.1 Performance Evaluation

We evaluated the running time of our algorithm using the synthetic dataset, to understand its performance with respect to various dataset sizes, in two different scenarios, an *insertion-only stream*, and a *fully dynamic* stream, both obtained from our synthetic dataset. As a comparative benchmark, we compare our approach with an online implementation of a "vanilla" LSH scheme (later *Vanilla-MH*), where profile sketches are computed online using 2-wise independent hash functions (that is also our signature scheme).

We tested two versions of our algorithm. The first version henceforth called *DynSymSearch (DSS)* maintains the sketches of Algorithm 1 and computes fingerprints only at query time. The second, called *DynSymSearch Proactive* (or simply *DSS Proactive*), instead maintains a set of fingerprints online, with every update (that is, after line 5 of Algorithm 1, reflecting the most recent change from the stream).

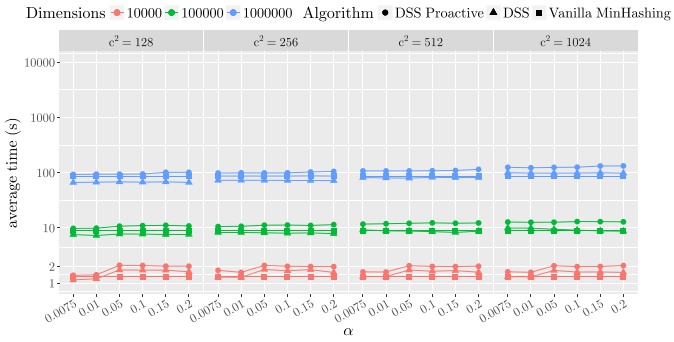
The choice of the first or the second implementation depends on the use case, with a trade-off between query responsiveness and additional space required for computing and storing the signatures of sets.

Let us now focus on the algorithms that update signatures online. When inserting item j added to set i, both *DSS Proactive* and *Vanilla-MH* behave in the same way. When an element is added, all hash-functions are evaluated on the new element, and updated in case such value is the new minimum. Let $k = lsb(h(j))$. In case of deletions, both will have to recompute signatures, yet while *Vanilla-MH* has to do so for the full user profile, *DSS Proactive* has to recompute signatures only for the two compressed bucket sets $T^i_{k,\bullet}$ and $T^i_{0,\bullet}$. A further optimization that we implemented in *DSS Proactive*, is the *selective recomputation* of signatures in case of deletions. In case of deletions of an item j, we recompute a set of signatures for $T^{(i)}_k$ only if the bucket is *sensitive*, i. e., its corresponding set cardinality and similarity threshold are such that k is the range specified by line 5 of Algorithm 2. This allows to ignore many costly recomputations.

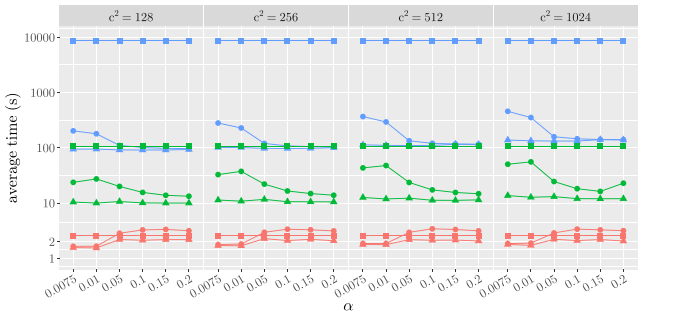
Now we can move on to comparing the three on the various settings. We set $\ell = 5$, $m = 40$ as amplifying parameters for signatures of all algorithms, and further set $r_1 = 0.5$ for our two algorithms. The choice of ℓ, m is not extremely important: indeed for the sake of runtime comparison all algorithms should only share the same "hashing-related" overhead. Average running times of 10 independent realizations of each algorithm are plotted in Fig. 1 where we study the impact of the parameters.

The running time of our algorithms is influenced by their parameters to different extents. In particular, the number of buckets c^2 has impact on both our algorithms (especially for *DSS Proactive*) as it directly implies more hash function evaluation for fingerprints.

For the insertion-only stream (Fig. 1a), we see that the performances of the three algorithms are somewhat comparable, which is expected, considering that *Vanilla-MH* is to some extent naturally contained in both versions of our algorithm. In *DSS* they are computed only at query time on the sensitive sketches, rendering it the fastest option for this scenario.

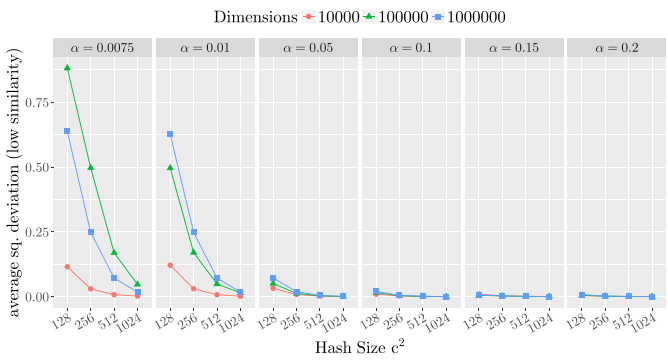


(a) Insertion only stream

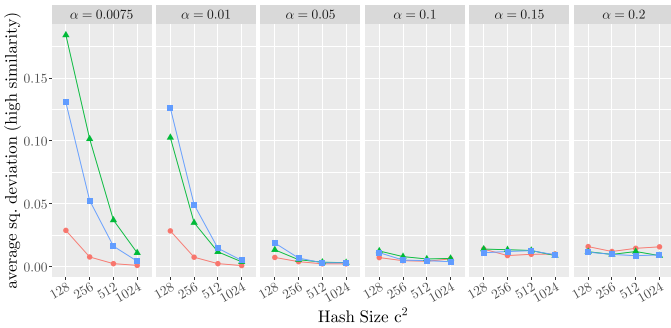


(b) Fully dynamic stream

Fig. 1. [*Synthetic*] Running time of our algorithms compared to a 2-wise MinHashing based LSH implementation, in insertion-only or fully dynamic setting for different values of $|U|$. y-axes are in log-scale. The summary running times are the mean values of 10 repetitions. In the *insertion only* setting (a), all algorithms have comparable performance, while in the *dynamic* setting (b), Vanilla MH takes from 12 to 100 times more time than the two variants of our algorithm, due to extensive recomputation of signatures in case of deletions.



(a) Low similarity pairs



(b) High similarity pairs

Fig. 2. [*Synthetic*] Average squared deviation for high similarity ($J \geq 0.2$) and low similarity ($J < 0.2$) pairs in the synthetic dataset, for various parameter choices. At inverse sampling rate $\alpha = 0.1$, the error for both high similarity and low similarity pairs was below 0.05, even for a very small number of buckets ($c^2 = 128$).

When considering deletions, things change dramatically. As can be seen in Fig. 1b, deletions represent a problem for both *Vanilla-MH* and *DSS Proactive*: as the fingerprint computation is not reversible, after a deletion they must all be consistently recomputed. However, our algorithm is less affected: thanks to its compression and cardinality-based bucketing system, the updates are, to some extent, more local, as they impact only the sensitive buckets. We note that *DSS Proactive* has some values of α where the running time increases: these values allow for a wider range of buckets to become sensitive as long as the set cardinalities vary with the stream, implying more signature recomputations when each k becomes queryable. Overall, *DSS* is consistently faster then the other two options, while the performance of *Vanilla-MH* is very poor, taking from 12 to 100 times more time than *DSS Proactive*, for $d = 1M$. We also remark that in the implementation of *Vanilla-MH*, we are forced to store the entire data set in order to deal with deletions, to be able to recompute the fingerprints. This requirement is indeed not feasible in many settings. Furthermore, even for these comparatively sparse data sets, our algorithm has significant space savings.

Quality of Approximation. We now move to examine the quality of approximation of our algorithm (which is the same for both online and offline implementations), on the synthetic dataset, as a function of our two main parameters, α and c^2. Concerning α, there are two opposite cases. If the inverse sampling rate is too low, we might have chosen set representative buckets with many samples: this means high chance of collisions which decreases the approximation ratio. On the other hand if it is too high, the selected set of items might not be sensitive. A higher bucket size instead, always means less collisions, for an increased space occupation of the sketches.

Fig. 2 shows the values of the average squared deviation of the sketched similarities obtained with our algorithm, and their exact Jaccard similarity, on the synthetic dataset, for different value of d, and various values of the parameters α and c^2.

The goodness of a given α depends on the similarity of a pair in question. We show separate plots for high and low similarity pairs, that is pairs with Jaccard similarity respectively below and above 0.2. Their behavior is affected in a different way. First, low pairs tend to have higher average squared deviation than high pairs, this is expected as out sketches can better approximate high similarity pairs. Also, for both kind of pairs the distortion decreases with increasing c^2, independently of α as the number of collisions decrease monotonically. All deviations reach almost zero already at $\alpha = 0.05$ for all bucket sizes. For α above 0.1 we see that the deviation of high similarity pairs depart from the others, and especially for higher dimensional datasets tend to be slightly more distorted. Except for the lowest number of buckets, the average total deviation for these parameters was always below 0.1 and further decreased reaching to zero for larger bucket sizes. We note that these values of c^2 are below the theoretical bounds of Theorem 6, while having little to acceptable deviation for appropriately chosen values of α.

TABLE 2
Fraction of Pairs Reported as Candidates versus Best Number of False Negatives Given by our Algorithm for Various Choices of a and b

ℓ	m	% candidate pairs	False negatives
4	400	0.099	1393
5	50	0.016	5171
	150	0.028	2406
	300	**0.036**	**781**
	320	0.029	1587
6	200	0.0187	5415

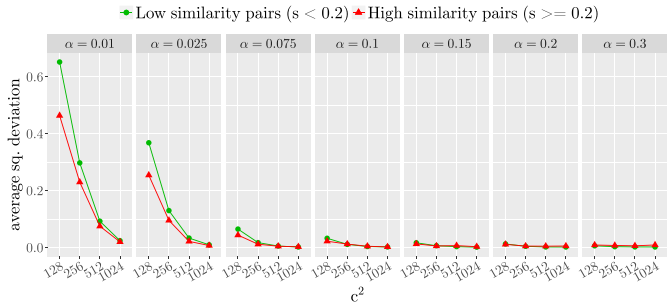
6.2 Analysis of the Last.fm Dataset

A realistic context like the one of Last.FM dataset, offers a valid playground to explore the performance of our similarity search. We use our algorithms to find high similarity pairs, to provide recommendations. We also compute a visualization of the most related user pairs, which illustrates an application of the sketching techniques from Section 4 to implicitly store an approximate distance matrix in small space. We note that the data is very sparse. Since the Jaccard index is highly sensitive to the support of the vectors, using it for this type of recommendation is more appropriate compared to other similarity measures such as Hamming, or cosine similarity.

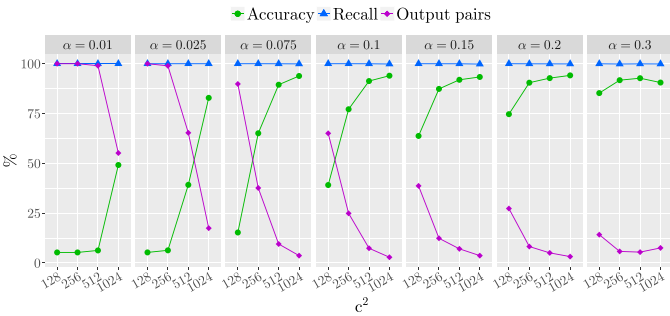
In Algorithm 2, we fixed $r_1 = 0.25$, therefore for each set A associated to each user profile, we added a min-hash value to H_k with $k \in [s-2, s] \cap I$, with $I = \{0, \log_2 \frac{1}{r_1}, \ldots\} = \{0, 2, 4, \ldots, \log_2 |U|\}$ (lines 3-4), where s is the actual cardinality of A at the time of the query (which we perform at the end of the stream). At the output of the filtering phase, we evaluate the similarity between users of a candidate pair using $k = \log_2(\alpha \cdot r_1 \cdot \max(|A|, |B|)$ and output $S(A_k, B_k)$. Note that this choice of k satisfies the first condition from Lemma 1. We remark that this dataset, as witnessed by the huge presence of very low similarity pairs (see Table 1), and very few pairs with higher similarity, is a challenge for any LSH scheme, as providing a good filtering behavior with low similarity thresholds requires many hash functions.

We performed multiple experiments in order to choose good parameters of ℓ and m to achieve a good filtering. We set a threshold on the maximum number of hash functions to use to 1600 hash functions. Then we also set a threshold on the maximum fraction of pairs that we accept to be reported as candidate pairs, to 10 percent. Then we tested a number of combinations of ℓ and m that are compatible with the similarity threshold r and meet our constraints, and report them in Table 2. The combination of $\ell = 5$ and $m = 300$ shows the lowest number of false negatives, and achieves a very good filtering, reporting only as low as 3.6 percent of pairs. We choose these values as amplification parameters for the filtering phase, and are fixed for all the experiments on this dataset.

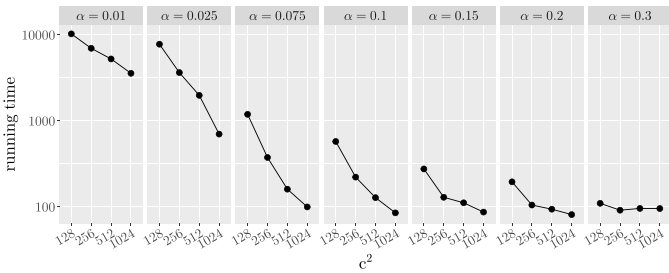
Fig. 3a shows average squared deviation values of the sketched similiarities obtained with our algorithm and their exact Jaccard similarity, as function of α and c^2. Like in Fig. 2, we show separate curves for pairs with Jaccard similarity below 0.2 (green curve) and high pairs (red curve). The same considerations made for the synthetic dataset hold, while we note that, for this dataset, the approximations of high similar pairs for very low bucket sizes appear slightly worse, possibly



(a) Average squared deviation



(b) Accuracy, Recall and Fraction of output pairs



(c) Running time (log-scale)

Fig. 3. [*Last.fm*] Approximation quality, Accuracy, Recall, Fraction of pairs found, and running time, for various combinations of the parameters.

because indeed the majority of them have a similarity value is closer to the threshold, with respect to the synthetic dataset. However, for appropriate values of the parameters, all deviations tend to zero. Fig. 3b shows other information regarding the detection performance of our filtering scheme. Recall that the sensitivity of our scheme is defined using Indyk and Motwani [11] kind of sensitivity, that is characterized by two different thresholds $r_2 < r_1$ (and corresponding regimes, with different approximation bounds as per Theorem 6). As a rule of thumb, r_2 is roughly of the order $r_1^{1.5}$, so we tolerate to report pairs with similarity above $r_2 = 0.125$, and consider this range as *true positives* (*TP*), *true negatives* (*TN*) pairs below r_2 that are correctly not reported. Conversely, pairs below r_2 that are reported as candidates by our algorithm are *false positives* (*FP*), and we consider *false negatives* (*FN*) pairs that are above the real threshold $r_1 = 0.25$ but were not reported. Fig. 3b shows values of $Accuracy = \frac{(\mathbf{TP}+\mathbf{TN})}{(\mathbf{TP}+\mathbf{TN}+\mathbf{FP}+\mathbf{FN})}$, $Recall = \frac{(\mathbf{TP})}{(\mathbf{TP}+\mathbf{FN})}$ and fraction of candidate pairs reported. We can notice that the recall is approximately 1 for all values of the parameters. Accuracy instead, increases for increasing c^2, as expected, and also for increasing α, until it deteriorates for very high values, like it was for the high similarity pairs in Fig. 3a. We notice that we get filtering above 90 percent

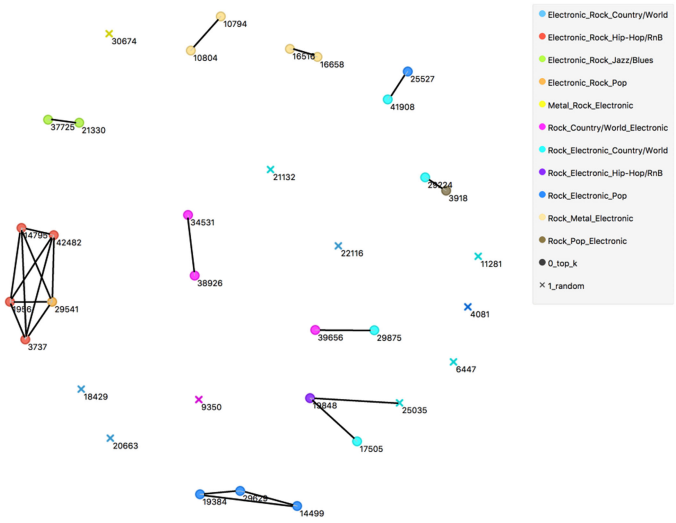


Fig. 4. $2D$ embedding of the profiles of 30 users (*top* and *random*), obtained running a MDS algorithm on the characteristic vectors. Circles are used to represent users belonging to top similar pairs ($\mathrm{sim} \geq 0.4$), and other 10 users were selected at random, and are marked with a cross symbol. Colors represent combinations of the top three user genres scores (see Section 6.2.1 for further details).

starting from $\alpha = 0.075$, for 512 buckets. Lastly, in Fig. 3c we see that we achieve very small running times from $\alpha = 0.075$, as a consequence of the filtering. We remark that these plots show the performances of the filtering algorithm alone without any further pruning step. Yet, as reported by our very low deviation from actual similarities, we note that when completely avoiding false negatives is of primary concern, one can decide to choose a lower r_1 (and/or a different l, m combination) to retain more pairs in the candidate selection phase, and then perform another linear filtering using the accurate estimation given by our sketches.

6.2.1 Visualizing top Similar Users

We conclude showing a visualization of the most similar Last.fm users found by *DynSymSearch*. For a predefined order of the elements in U, that is, our collection of music artists, we can view user profiles as their characteristic binary vectors, where an entry is 1 at a given time if a given user has recently listened to the corresponding artist. Given the high dimensionality of U, it is very hard to find a way to make sense of such similarities. We have taken various steps to reach the following two objectives: i) find a lower dimensional representation (ideally 2D points) of the user profiles that can mostly retain their Jaccard similarities, and ii) enrich such points with lower resolution information that helps to visually distinguish similar pairs without recurring to artist annotation.

Our input is a set of characteristic vectors, representing profiles of a set S of 34 users, 24 of which form the top 14 similar pairs i. e., pairs with similarity above 0.4 (see Table 1) and other 10 users selected at random. We refer to the former users as the *top k* users, and call the latter users *random*.

For implementing step i) we resort to Multidimensional Scaling (MDS) [41], a technique that takes in input a matrix of pairwise distances (notably euclidean and Jaccard, among others) of an input set of objects, and finds an N-dimensional projection of points, such that the between-object distances are preserved as well as possible. Each object is then assigned coordinates in each of the N dimensions. We used a Python implementation of MDS from the Orange Data Mining library [42], where we set $N = 2$ and input a Jaccard distance matrix computed on all pairs of our 34 user characteristic vectors.

As per step ii) we used genre information from the original dataset in the form of a vector of scores for the music genres *Rock*, *Pop*, *Electronic*,*Metal*,*Hip-Hop/RnB*, *Jazz/Blues* and *Country/World*, where each entry is normalized so that their sum adds to 1. For each artist a appearing in some user profile, we derived a score vector computing the normalized sum of all score vectors of tracks authored by him, present in our dataset, and then in turn used the same mechanism for deriving a score for a user listening to a set of artists, determining a 7-dimensional vector, or *genre-based profile* for each user.

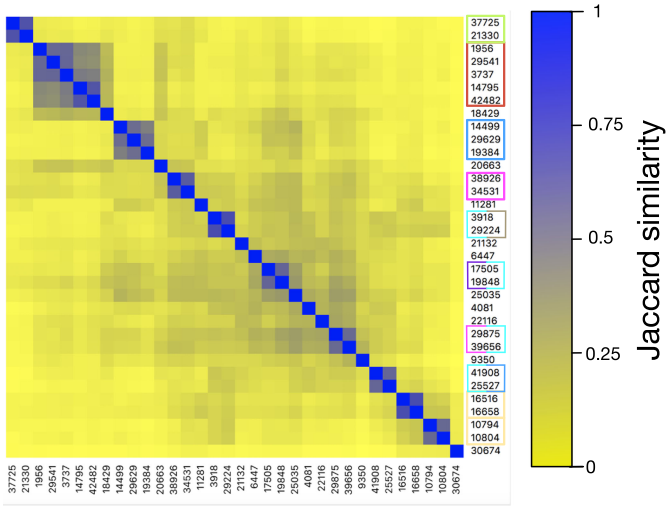
Fig. 4 depicts a result of the combinations of both steps: a 2-dimensional MDS visualization of S computed using artist-based Jaccard similarity, annotated with colors reflecting the first 3 entries by score as per the corresponding users genres-based profiles. Also, edges show pairs for which the Jaccard similarity is above the threshold 0.1 (note that more than 14 edges are reported, as some users are involved in mildly-similar pairs with other users from *top k*, yet with similarity lower than 0.4). We notice that the majority of users have *Rock* or *Electronic* among their main genres, this is a characteristic of the dataset [40].

Overall, a clustered structure becomes apparent when considering both distance and genre-based colors, (also, we see that *random* users — marked with a cross symbol — are mostly spread out and are not involved in any pairs). Some pairs of *top* users have different colors: this possibly means that their intersection involves a subset of such genres, which is quite natural.

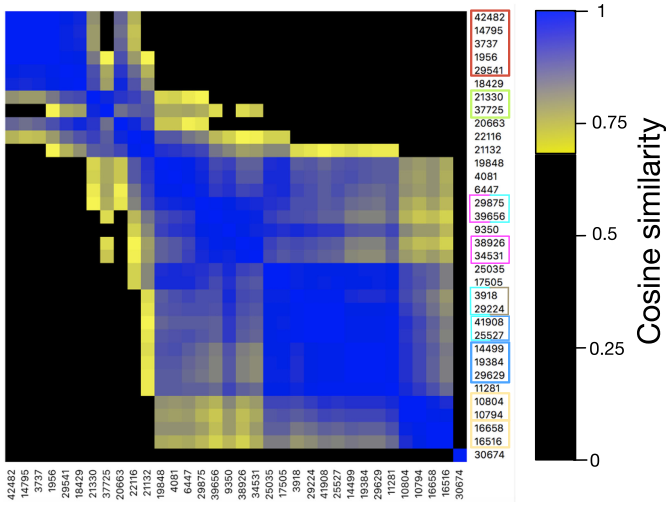
To complement Fig. 4, in Fig. 5, we show heatmaps of two similarity matrices, Jaccard similarities of artist-based profiles (Fig. 5a) and Cosine similarities of genre-based profiles (which span the range between 0 and 1, being vectors with only positive components), in Fig. 5b, arranged using the output of a hierarchical clustering algorithm with *ordered leaves* representation [43], i. e., maximizing the sum of similarities among adjacent elements. We see that the clustering structure is apparent, and preserved, in both matrices (see the colored boxes on the user ids), although way clearer in the Jaccard matrix. This is expected as artists-based profiles have a far more granular resolution, and are therefore sparser with respect to genre-based profiles, especially considering that the main genres are almost the same for all users. This is also a witness of the fact that artist information is more suitable to achieve real personalized recommendations than genres, which motivates our choice of artists as user profile features.

7 Conclusion

In this paper, we presented scalable approximation algorithms for Jaccard-based similarity search in dynamic data streams. Specifically, we showed how to sketch the Jaccard similarity via a black box reduction to ℓ_0 norm estimation, and we gave a locality-sensitive hashing scheme that quickly filters out low-similarity pairs. To the best of our knowledge, these are the first algorithms that can handle item deletions.



(a) Clustered Jaccard similarity matrix (users profiles)



(b) Clustered cosine similarity matrix (genres)

Fig. 5. Clustered similarity matrices for top and random users of Fig. 4. Plots show heatmap output of a hierarchical clustering algorithm using (a) Jaccard distance between users profiles (listened Last.fm artists). (b) Cosine distance genres-based profile. In both figures the colored boxes highlight the clusters present in Fig. 4.

In addition to theoretical guarantees, we showed that the algorithm has competitive running times to the established min-hashing approaches. We also have reason to believe that the algorithm can be successfully applied in real-world applications, as evidenced by its performance for finding Last.fm users with similar musical tastes.

It would be interesting to extend these ideas for other similarity measures. Though we focused mainly on the Jaccard index, our approach works for any set-based similarity measure supporting an LSH, compare the characterization of Chierichetti and Kumar [37]. It is unclear whether our techniques may be reused for other similarities applied in collaborative filtering, such as the Kendall-tau metric.

Future work might also focus on collecting data sets with insertions and deletions. Even streaming benchmarks typically consist only of a final data set and are therefore inherently insertion-only. We feel that a formal model for capturing dynamic data will be of considerable value both for the designing and evaluating algorithms.

Acknowledgments

C.S. is supported by ERC Advanced Grant 788893 AMDROMA. A preliminary version of this paper appeared in WSDM 2018 [1].

References

[1] M. Bury, C. Schwiegelshohn, and M. Sorella, "Sketch 'em all: Fast approximate similarity search for dynamic data streams," in *Proc. 11th ACM Int. Conf. Web Search Data Mining*, 2018, pp. 72–80.
[2] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, "Syntactic clustering of the web," *Comput. Netw.*, vol. 29, no. 8–13, pp. 1157–1166, 1997.
[3] S. Guha, R. Rastogi, and K. Shim, "ROCK: A robust clustering algorithm for categorical attributes," *Inf. Syst.*, vol. 25, no. 5, pp. 345–366, 2000.
[4] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang, "Finding interesting associations without support pruning," *IEEE Trans. Knowl. Data Eng.*, vol. 13, no. 1, pp. 64–78, Jan./Feb. 2001.
[5] Y. Shi, M. Larson, and A. Hanjalic, "Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges," *ACM Comput. Surv.*, vol. 47, no. 1, Art. no. 3.
[6] A. Das, M. Datar, A. Garg, and S. Rajaram, "Google news personalization: Scalable online collaborative filtering," in *Proc. 16th Int. Conf. World Wide Web*, 2007, pp. 271–280.
[7] P. Indyk and T. Wagner, "Near-optimal (euclidean) metric compression," in *Proc. 28th Annual ACM-SIAM Symp. Discrete Algorithms*, 2017, pp. 710–723.
[8] R. Pagh, M. Stöckel, and D. P. Woodruff, "Is min-wise hashing optimal for summarizing set intersection?" in *Proc. Proc. 33rd ACM SIGMOD-SIGACT-SIGART Symp. Principles Database Syst.*, 2014, pp. 109–120.
[9] S. Janssens, "Bell inequalities in cardinality-based similarity measurement," PhD dissertation, Ghent Univ., Gent, Belgium, 2006.
[10] M. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proc. 34th Annu. ACM Symp. Theory Comput.*, 2002, pp. 380–388.
[11] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. 30th Annu. ACM Symp. Theory Comput.*, 1998, pp. 604–613.
[12] A. Z. Broder, "On the resemblance and containment of documents," in *Proc. Compression Complexity Sequences 1997*, 1997, Art. no. 21.
[13] A. Z. Broder, "Identifying and filtering near-duplicate documents," in *Proc. 11th Annu. Symp. Combinatorial Pattern Matching*, 2000, pp. 1–10.
[14] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," *J. Comput. Syst. Sci.*, vol. 60, no. 3, pp. 630–659, 2000.
[15] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Datasets, 2nd Ed.* Cambridge, U.K.: Cambridge Univ. Press, 2014.
[16] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Commun. ACM*, vol. 51, no. 1, pp. 117–122, 2008.
[17] E. Cohen and H. Kaplan, "Summarizing data using bottom-k sketches," in *Proc. 26th Annu. ACM Symp. Principles Distrib. Comput.*, 2007, pp. 225–234.
[18] E. Cohen and H. Kaplan, "Bottom-k sketches: Better and more efficient estimation of aggregates," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, 2007, pp. 353–354.
[19] J. M. Hellerstein, P. J. Haas, and H. J. Wang, "Online aggregation," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1997, pp. 171–182.
[20] P. Indyk, "A small approximately min-wise independent family of hash functions," *J. Algorithms*, vol. 38, no. 1, pp. 84–90, 2001.
[21] G. Feigenblat, E. Porat, and A. Shiftan, "Exponential time improvement for min-wise based algorithms," in *Proc. 22nd Annu. ACM-SIAM Symp. Discrete Algorithms*, 2011, pp. 57–66.
[22] M. Dietzfelbinger, "Universal hashing and k-wise independent random variables via integer arithmetic without primes," in *Proc. 13th Annu. Symp. Theoretical Aspects Comput. Sci.*, 1996, pp. 569–580.
[23] K. Chung, M. Mitzenmacher, and S. P. Vadhan, "Why simple hash functions work: Exploiting the entropy in a data stream," *Theory Comput.*, vol. 9, pp. 897–945, 2013.

[24] M. Thorup, "Bottom-k and priority sampling, set similarity and subset sums with minimal independence," in *Proc. 45th Annu. ACM Symp. Theory Comput.*, 2013, pp. 371–380.
[25] M. Patrascu and M. Thorup, "Twisted tabulation hashing," in *Proc 24th Annu. ACM-SIAM Symp. Discrete Algorithms*, 2013, pp. 209–228.
[26] M. Thorup, "Simple tabulation, fast expanders, double tabulation, and high independence," in *Proc. IEEE 54th Annu. Symp. Found. Comput. Sci.*, 2013, pp. 90–99.
[27] A. Pagh, R. Pagh, and M. Ruzic, "Linear probing with 5-wise independence," *SIAM Rev.*, vol. 53, no. 3, pp. 547–558, 2011.
[28] M. Patrascu and M. Thorup, "On the *k*-independence required by linear probing and minwise independence," *ACM Trans. Algorithms*, vol. 12, no. 1, pp. 8:1–8:27, 2016.
[29] S. Dahlgaard, M. B. T. Knudsen, and M. Thorup, "Fast similarity sketching," in *Proc. IEEE 58th Annu. Symp. Found. Comput. Sci.*, 2017, pp. 663–671.
[30] P. Li, A. B. Owen, and C. Zhang, "One permutation hashing," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst. - Vol. 2*, 2012, pp. 3122–3130.
[31] A. Shrivastava and P. Li, "Densifying one permutation hashing via rotation for fast near neighbor search," in *Proc. Proc. 31st Int. Conf. Int. Conf. Mach. Learn. - Vol. 32*, 2014, pp. 557–565.
[32] S. Dahlgaard, M. B. T. Knudsen, and M. Thorup, "Practical hash functions for similarity estimation and dimensionality reduction," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 6618–6628.
[33] K. Q. Weinberger, A. Dasgupta, J. Langford, A. J. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 1113–1120.
[34] Y. Bachrach and E. Porat, "Fingerprints for highly similar streams," *Inf. Comput.*, vol. 244, pp. 113–121, 2015.
[35] Y. Bachrach and E. Porat, "Sketching for big data recommender systems using fast pseudo-random fingerprints," in *Proc. Int. Colloquium Automata Lang. Program.*, 2013, pp. 459–471.
[36] P. Li and A. C. König, "Theory and applications of *b*-bit minwise hashing," *Commun. ACM*, vol. 54, no. 8, pp. 101–109, 2011.
[37] F. Chierichetti and R. Kumar, "LSH-preserving functions and their applications," *J. ACM*, vol. 62, no. 5, 2015, Art. no. 33.
[38] D. M. Kane, J. Nelson, and D. P. Woodruff, "An optimal algorithm for the distinct elements problem," in *Proc. 29th ACM SIGMOD-SIGACT-SIGART Symp. Principles Database Syst.*, 2010, pp. 41–52.
[39] L. Naish, H. J. Lee, and K. Ramamohanarao, "A model for spectra-based software diagnosis," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, 2011, Art. no. 11.
[40] A. Anagnostopoulos and M. Sorella, "Learning a macroscopic model of cultural dynamics," in *Proc. Int. Conf. Data Mining*, 2015, pp. 685–690.
[41] F. Wickelmaier, "An introduction to mds," *Sound Quality Res. Unit*, Aalborg University, Aalborg, Denmark, vol. 46, no. 5, pp. 1–26, 2003.
[42] J. Demšar, T. Curk, A. Erjavec, Črt Gorup, T. Hočevar, M. Milutinovič, M. Možina, M. Polajnar, M. Toplak, A. Starič, M. Štajdohar, L. Umek, L. Žagar, J. Žbontar, M. Žitnik, and B. Zupan, "Orange: Data mining toolbox in python," *J. Mach. Learn. Res.*, vol. 14, pp. 2349–2353, 2013.
[43] Z. Bar-Joseph, D. K. Gifford, and T. S. Jaakkola, "Fast optimal leaf ordering for hierarchical clustering," *Bioinf.*, vol. 17, no. suppl_1, pp. S22–S29, 2001.

Marc Bury received the PhD degree in computer science from the Technical University of Dortmund, in 2016. He now works with Google Zürich. His past work revolved mostly on complexity theory, in particular BDDs, and streaming algorithms. His current research focuses on scalable algorithms for web-based applications.

Chris Schwiegelshohn received the PhD degree in computer science from the Technical University of Dortmund, in 2017. He is currently an assistant professor with Sapienza, University of Rome. His main research interests cover streaming, dynamic algorithms, and learning problems such as clustering.

Mara Sorella received the PhD degree in computer science and engineering from Sapienza, University of Rome, in 2018. She continues to work there as a postdoc. Her current interests include distributed computing, security, and data mining.