58th Annual IEEE Symposium on Foundations of Computer Science

Fast Similarity Sketching

Søren Dahlgaard, Mathias Bæk Tejs Knudsen, Mikkel Thorup
Department of Computer Science
University of Copenhagen
Copenhagen, Denmark
Email: soren.dahlgaard@gmail.com, mathias@tejs.dk, mikkel2thorup@gmail.com

***Abstract*—We consider the *Similarity Sketching* problem: Given a universe $[u] = \{0, \dots, u-1\}$ we want a random function S mapping subsets $A \subseteq [u]$ into vectors $S(A)$ of size t, such that similarity is preserved. More precisely: Given sets $A, B \subseteq [u]$, define $X_i = [S(A)[i] = S(B)[i]]$ and $X = \sum_{i \in [t]} X_i$. We want to have $\mathrm{E}[X] = t \cdot J(A, B)$, where $J(A, B) = |A \cap B| / |A \cup B|$ and furthermore to have strong concentration guarantees (i.e. Chernoff-style bounds) for X. This is a fundamental problem which has found numerous applications in data mining, large-scale classification, computer vision, similarity search, etc. via the classic MinHash algorithm. The vectors $S(A)$ are also called *sketches*.**

The seminal $t\times$*MinHash* algorithm uses t random hash functions $h_1, \dots, h_t$, and stores $(\min_{a \in A} h_1(A), \dots, \min_{a \in A} h_t(A))$ as the sketch of A. The main drawback of MinHash is, however, its $O(t \cdot |A|)$ running time, and finding a sketch with similar properties and faster running time has been the subject of several papers. Addressing this, Li et al. [NIPS'12] introduced *one permutation hashing (OPH)*, which creates a sketch of size t in $O(t + |A|)$ time, but with the drawback that possibly some of the t entries are "empty" when $|A| = O(t)$. One could argue that sketching is not necessary in this case, however the desire in most applications is to have *one* sketching procedure that works for sets of all sizes. Therefore, filling out these empty entries is the subject of several follow-up papers initiated by Shrivastava and Li [ICML'14]. However, these "densification" schemes fail to provide good concentration bounds exactly in the case $|A| = O(t)$, where they are needed.

In this paper we present a new sketch which obtains essentially the best of both worlds. That is, a fast $O(t \log t + |A|)$ expected running time while getting the same strong concentration bounds as MinHash. Our new sketch can be seen as a mix between sampling with replacement and sampling without replacement. We demonstrate the power of our new sketch by considering popular applications in large-scale classification with linear SVM as introduced by Li et al. [NIPS'11] as well as approximate similarity search using the LSH framework of Indyk and Motwani [STOC'98]. In particular, for the j_1, j_2-approximate similarity search problem on a collection of n sets we obtain a data-structure with space usage $O(n^{1+\rho} + \sum_{A \in \mathcal{C}} |A|)$ and $O(n^\rho \log n + |Q|)$ expected time for querying a set Q compared to a $O(n^\rho \log n \cdot |Q|)$ expected query time of the classic result of Indyk and Motwani.

***Keywords*-hashing; similarity sketch; sketching; Jaccard similarity; LSH; locality-sensitive hashing**

I. Introduction

In this paper we consider the following problem which we call the *similarity sketching* problem. Given a large universe $[u] = \{0, \dots, u-1\}$ and positive integer t we want a random function S mapping subsets $A \subseteq [u]$ into vectors (which we will call sketches) $S(A)$ of size t, such that similarity is preserved. More precisely, given sets $A, B \subseteq [u]$, define $X_i = [S(A)[i] = S(B)[i]]$ for each $i \in [t]$, where $S(A)[i]$ denotes the ith entry of the vector $S(A)$ and $[x]$ is the Iverson bracket notation with $[x] = 1$ when x is true and 0 otherwise. Let $X = \sum_{i \in [t]} X_i$, then we want $\mathrm{E}[X] = t \cdot J(A, B)$, where $J(A, B) = |A \cap B| / |A \cup B|$ is the *Jaccard similarity* of A and B. That is, the sketches can be used to estimate $J(A, B)$ by doing a pair-wise comparison of the entries. We will call this the *alignment property* of the similarity sketch. Finally, we want to have Chernoff-style concentration bounds on the value of X. The standard solution to this problem is $t\times$MinHash algorithm[1]. The algorithm works as follows: Let $h_0, \dots, h_{t-1} : [u] \to [0, 1]$ be random hash functions and define $S(A) = (\min_{a \in A} h_0(a), \dots, \min_{a \in A} h_{t-1}(a))$. This corresponds to sampling t elements from A with replacement and thus has all the above desired properties.

MinHash was originally introduced by Broder et al. [5], [6] for the AltaVista search engine and has since been used as a standard tool in many applications including duplicate detection [6], [9], all-pairs similarity [4], large-scale learning [14], computer vision [17], and similarity search [11]. The main motivation for hashing-based approaches to these problems is the continuing increases in dimensionality of modern datasets. Weinberger et al. [23] considered sets from a universe of size 16 trillion ($u \approx 10^{13}$) and Tong [22] considered sets with $u \approx 10^9$. Furthermore, when working with text, input is often represented by w-shingles (i.e. w contiguous words) with $w \geqslant 5$. This further increases the dimension from, say roughly 10^5 common english words to $u \approx 10^{5w}$.

The main drawback of MinHash is, however, the $O(t \cdot |A|)$ running time. For practical applications, where the data is ultra high dimensional, this sketch creation time is often a bottleneck. As an example, [14] suggests using $t = 500$ and [12] suggests using $t = 4000$. Several papers have therefore been concerned with finding a similarity sketch with equal power and faster running time.

[1] https://en.wikipedia.org/wiki/MinHash

Bachrach and Porat [3] suggested a more efficient way of maintaining t MinHash values with t different hash functions. They use t different polynomial hash functions that are related, yet pairwise independent, so that they can systematically maintain the MinHash for all t polynomials in $O(\log t)$ time per element of A. There are two issues with this approach: It is specialized to work with polynomials and MinHash is known to have constant bias unless the polynomials considered have super-constant degree [15], and this bias does not decay with independent repetitions. Also, because the experiments are only pairwise independent, the concentration is only limited by Chebyshev's inequality and thus nowhere near the Chernoff bounds we want for many applications.

Another direction introduced by Li et al. [13] is *one permutation hashing* (OPH) which works by hashing the elements of A into t buckets and performing a MinHash in each bucket using the same hash function. While this procedure gives $O(t + |A|)$ sketch creation time it also may create empty buckets and thus only obtains a sketch with $t' \leqslant t$ entries when $|A| = o(t \log t)$. One may argue that sketching is not even needed in this case. However, a common goal in applications of similarity sketching is to have *one* sketching procedure which works for all set size – eg. one data structure that works for an entire data set of different sizes in the case of approximate similarity search. It is thus very desirable that the similarity sketch works well independently of the size of the input set.

Motivated by this, several follow-up papers [20], [19], [18] have tried to give different schemes for filling out the empty entries of the OPH sketch ("densifying" the sketch). These papers all consider different ways of copying from the full entries of the sketch into the empty ones. Due to this approach, however, these densification schemes all fail to give good concentration guarantees when $|A|$ is small, which is exactly the cases in which OPH gives many empty bins and densification is needed. This is because of the fundamental problem that unfortunate collisions in the first round cannot be resolved in the second round when copying from the full bins. To understand this consider the following extreme example: Let A be a set with two elements. Then with probability $1/t$ these two elements end in the same bin, and after copying the entire densified sketch ends up consisting of just one element. This leads to very poor similarity estimation. This behaviour is illustrated with experiments in Figure 1. Furthermore, the state-of-the-art densification scheme of Shrivastava [18] has an expected running time of $O(|A| + t^2/|A|)$ and thus potentially takes $O(t^2)$ time when $|A|$ is small.

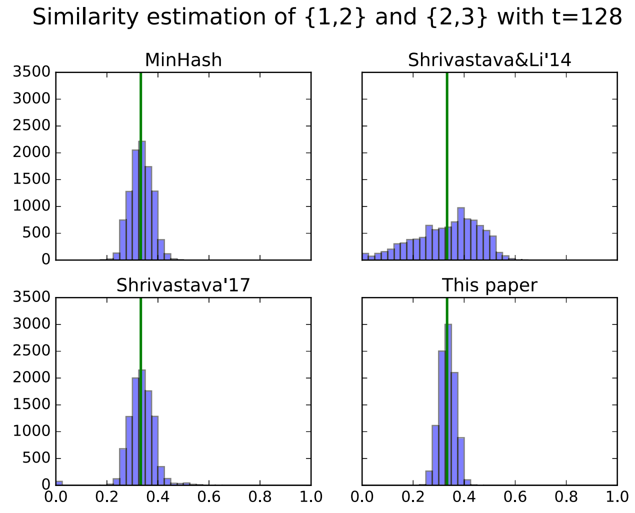

Figure 1. Experimental evaluation of similarity estimation of the sets $A = \{1, 2\}$ and $B = \{2, 3\}$ with different similarity sketches and $t = 128$. Each experiment is repeated 10000 times and the y-axis reports the frequency of each estimate. The green line indicates the actual similarity. The two methods based on OPH do not give as strong concentration as the others. Note that for the method of Shrivastava [18] has several estimates of similarity zero, which never occurs with MinHash or our method. The poor concentration is because each set has a probability of $1/t$ to be a single-element sketch. Our new method outperforms MinHash as it has an element of "without replacement".

A. Our contribution

In this paper we obtain a sketch which essentially obtains the best of both worlds. That is, strong concentration guarantees for similarity estimation as well as a fast expected sketch creation time of $O(t \log t + |A|)$. Our new sketch can be seen as a mixture between sampling with and without replacement and in many cases outperforms MinHash. An example of this can be seen in the toy example of Figure 1, where the "without replacement"-part of our sketch gives better concentration compared to MinHash. Our sketch can be directly employed in any place where $t\times$MinHash is currently employed to improve the running time. In this paper we focus on two popular applications, which are large-scale learning with linear SVM and approximate similarity search with LSH. We describe these applications in more detail below.

Another strength of our new sketch is that it can be implemented using just one mixed tabulation hash function (introduced by Dahlgaard et al. [8]) which can be evaluated in $O(1)$ time.

Theorem 1. *Let $[u] = \{0, 1, 2, \ldots, u-1\}$ be a set of keys and let t be a positive integer. There exists an algorithm that given a set $A \subseteq [u]$ in expected time $O(|A| + t \log t)$ creates a size-t vector $v(A)$ of non-negative real numbers with the following properties. For two sets $A, B \subseteq [u]$ it holds that $v(A \cup B)_i = \min\{v(A)_i, v(B)_i\}$ for each index $i \in [t]$. For $i \in [t]$ let $X_i = 1$ if $v(A)_i = v(B)_i$ and 0 otherwise and let $X = \frac{1}{t}\sum_{i\in[t]} X_i$. Then $E[X] = J$ where*

$J = J(A, B)$ *and for* $\delta > 0$ *it holds that:*

$$\Pr[X \geqslant J(1+\delta)] \leqslant \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^t ,$$
$$\Pr[X \leqslant J(1-\delta)] \leqslant \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^t .$$

We also present a way to compute our sketch in a streaming context with essentially the same running time.

Large-scale learning: Li et al. [14] considered using similarity sketching for applications in large-scale linear learning. In particular they showed how to naturally integrate MinHash into linear SVM and logistic regression to avoid computations on extremely large data points. The idea is as follows: For each input set A, they create a $t\times$MinHash similarity sketch and truncate each value in the sketch to b bits (called *b-bit minwise hashing*). They then create a vector of size $2^b \cdot t$ by concatenating the indicator vectors (of size 2^b) for each entry in the b-bit similarity sketch. By the alignment property of the similarity sketch it follows that the Jaccard similarity of two sets can be estimated as the dot-product of the two corresponding size-$2^b t$ vectors (with a bias depending on b). This is exactly the property needed by a linear SVM in order to perform efficient classification. As the linear SVM performs classification using a single dot-product the classification time then becomes $O(2^b t + t \cdot |A|)$ when using $t\times$MinHash. Using our new similarity sketch we immediately improve this to $O((2^b + \log t) \cdot t + |A|)$ which removes a major bottleneck (see [13]).

We note that it is crucial to this application that the similarity sketch satisfies the *alignment property* as also noted by Li et al. [13], as the similarity estimation can otherwise not be implemented with a dot-product.

Speeding up LSH: One of the most popular applications of the MinHash algorithm is the *approximate similarity search problem*. Here, we are given a collection, $\mathcal{C}$, of n sets from some universe $[u]$ as well as two parameters $0 \leqslant j_2 < j_1 \leqslant 1$. The task is to pre-process $\mathcal{C}$ such that given a query set $Q \subseteq [u]$ we can efficiently return a set $A \in \mathcal{C}$ with $J(A, Q) \geqslant j_2$ if there exists some $B \in \mathcal{C}$ with $J(B, Q) \geqslant j_1$. It is common to assume that j_1, j_2 are constants and we do the same in this paper. To address this problem, Indyk and Motwani [11] introduced the *Locality Sensitive Hashing* (LSH) framework. For parameters L, K, they created L different $K\times$MinHash sketches, $S_0(A), \ldots, S_{L-1}(A)$ for each set $A \in \mathcal{C}$. A query is then answered by computing L different $K\times$MinHash sketches $S_0(Q), \ldots, S_{L-1}(Q)$ for Q and for each $i \in [L]$ comparing Q to each set $A \in \mathcal{C}$ with $S_i(A) = S_i(Q)$. This gives a total space usage of $O(L \cdot n + \sum_{A\in\mathcal{C}} |A|)$ and an expected query time of $O(L \cdot K \cdot |Q|)$. By carefully choosing L and K they obtain a space usage of $O(n^{1+\rho} + \sum_{A\in\mathcal{C}} |A|)$ and expected query time of $O(|Q| \cdot n^{\rho} \log n)$, where $\rho = \log(1/j_1)/\log(1/j_2)$.

Following this seminal work it has become practice to evaluate algorithms in terms of their ρ-value, and several papers are concerned with reducing this value (see e.g. [1], [2], [7]) using increasingly sophisticated methods based on eg. *data-dependant hashing* as in [1], [2]. Using the LSH framework of [11] the query time is dominated by two parts: 1) The data structure returns $O(L)$ expected "false positives" which have to be filtered out in roughly $O(L \cdot |Q|)$ time, and 2) we have to compute $O(L \cdot K)$ hash values for the similarity sketches giving $O(L \cdot K \cdot |Q|)$ time when using MinHash. One way to remove this multiplicative dependance on Q is by using an "intermediate" similarity sketch of size $O(\log^3 n)$ and generating the similarity sketches of the LSH structure by sampling directly from this vector. This gives an expected query time of $O((L + |Q|) \cdot \log^3 n)$. This is still very time consuming when $|Q|$ and n are large, and thus removing the multiplicative dependance on $|Q|$ without introducing a large polylogarithmic factor was the main motivation behind studying OPH densification schemes [19], [20], [18]. However, as mentioned earlier, these densification schemes do not give the concentration bounds necessary for the LSH analysis to work.

In this paper we address the above issue, speeding up the query time of the LSH framework. Building upon ideas from Henzinger and Thorup [10] and using the similarity sketch from this paper we give a method that filters out false positives of 1) above in expected constant time, however, the main work lies in dealing with 2). To improve this part we show that we can use our new similarity sketch as an intermediate vector and sample from this in a clever way to build the LSH table in $O(L \cdot K + |Q|)$ time thus improving the total query time to $O(L \cdot K + |Q|) = O(n^{\rho} \log n + |Q|)$ expected time.

B. Related work

An alternative to $t\times$MinHash sketching is the bottom-t sketches described in [5], [21]. The idea is to use the t smallest hash values of a set A instead, applying just one hash function. However, similar to OPH this does not give us a sketch of size t when $|A|$ is small. Furthermore, the estimation procedure becomes more complicated and the sketches do not satisfy the alignment property, which is necessary for many applications (see Section I-A above).

A recent advance in approximate similarity search by Christiani and Pagh [7] is the *ChosenPath* method, which obtains a ρ-value better than the one obtained with LSH and MinHash. However, the authors consider a different similarity measure called Braun-Blanquet similarity, and in order to obtain their result for Jaccard similarity they have to convert between the two. They therefore assume that the input sets all have the same size, ℓ, and the authors suggest[2] using an $\ell\times$MinHash sketch as preprocessing to obtain this. When ℓ is large this pre-processing step is a bottleneck that can be sped up with out new similarity sketch.

[2]Personal communication

C. Notation

For a real number x and an integer k we define $x^{\underline{k}} = x(x-1)(x-2)\dots(x-k+1)$. For an expression P we let $[P]$ denote the variable that is 1 if P is true and 0 otherwise. For a non-negative integer n we let $[n]$ denote the set $[n] = \{0, 1, 2, \dots, n-1\}$.

II. Fast Similarity Sketching

In this section we present our new sketching algorithm, which takes a set $A \subseteq [u]$ as input and produces a sketch $S(A, t)$ of size t. When t is clear from the context we may write just $S(A)$.

Our new similarity sketch is simple to describe: Let $h_0, \dots, h_{2t-1}$ be random hash functions such that for $i \in [t]$ we have $h_i : [u] \to [t] \times [i, i+1)$ and for $i \in \{t, \dots, 2t-1\}$ we have $h_i : [u] \to \{i-t\} \times [i, i+1)$. For each hash function h_i we say that the output is split into a bin, b_i, and a value, v_i. That is, for $i \in [2t]$ and $a \in [u]$ we have $h_i(a) = (b_i(a), v_i(a))$, where $b_i(a)$ and $v_i(a)$ are restricted as described above. In particular, for $i \in \{t, \dots, 2t-1\}$ we have $b_i(a) = i - t$. We may then define the jth entry of the sketch $S(A)$ as follows:

$$S(A)[j] = \min\{v_i(a) \mid a \in A, i \in [2t], b_i(a) = j\} \ . \quad (1)$$

In particular, the hash functions $h_t, \dots, h_{2t-1}$ ensure that each entry of $S(A)$ is well-defined. Let i be the minimum index such that each bin is assigned an element using only $h_0, \dots, h_i$. Then, since since we have $v_i(a) < v_j(b)$ for any $a, b \in [u]$ and $0 \leqslant i < j < 2t$ it follows that our sketch can be computed using only the hash functions $h_0, \dots, h_i$, which allows for the efficient implementation defined in the algorithm of Figure 2. In this case we say that i is the *maximum hash index*.

```
Input: $A$, $t$, $h_0, \dots, h_{2t-1}$
Output: The sketch $S(A,t)$
 1: $S \leftarrow \infty^t$
 2: $c \leftarrow 0$
 3: for $i \in [2t]$ do
 4:     for $a \in A$ do
 5:         $b, v \leftarrow h_i(a)$
 6:         if $S[b] = \infty$ then
 7:             $c \leftarrow c + 1$
 8:         end if
 9:         $S[b] \leftarrow \min(S[b], v)$
10:     end for
11:     if $c = t$ then
12:         return $S$
13:     end if
14: end for
```

Figure 2. The `Fill-Sketch` procedure

We will start our analysis of $S(A)$ by bounding the running time of Figure 2.

Lemma 1. *Let $A \subseteq [u]$ be some set and let t be a positive integer. Then the expected running time of Figure 2 is $O(t \log t + |A|)$.*

Proof: We split the proof into two cases:

1) If $|A| \leqslant 2 \log t$ we have a trivial upper bound of $O(t \cdot |A|) = O(t \log t)$.
2) Otherwise, $|A| > 2 \log t$. Fix $i \in [t]$ to be the smallest value in such that $|A| \cdot i > 2 \cdot t \log t$. Then the probability of a given bin being empty after evaluating $h_0, \dots, h_{i-1}$ is at most

$$(1 - 1/t)^{|A| \cdot i} \leqslant (1 - 1/t)^{2 \cdot t \log t} \leqslant 1/t^2 \ .$$

It follows that the probability of any bin being empty is at most $1/t$ and thus the expected running time is $O(|A| \cdot i + \frac{|A| \cdot t}{t}) = O(t \log t + |A|)$.

■

Next, we will prove several properties of the sketch. The first is an observation that the sketch of the union of two sets can be computed solely from the sketches of the two sets.

Fact 1. *Let A, B be two sets and let t be a positive integer. Then*

$$S(A \cup B, t)[i] = \min(S(A, t)[i], S(B, t)[i]) \ .$$

The main technical lemma regarding the sketch is Lemma 2 below. Loosely speaking, the lemma bounds the kth moment of the sketch when estimating set similarity. We will use this lemma to show that we get an unbiased estimator as well as Chernoff-style concentration bounds.

Lemma 2. *Let A, B be sets with Jacard similarity $J(A, B) = J$ and let t be a positive integer. For each $i \in [t]$ let $X_i = [S(A,t)[i] = S(B,t)[i]]$. Let $I \subseteq [t]$ be a set of k indices. Then:*

$$E\left[\prod_{i \in I} X_i\right] \leqslant J^k \ ,$$

and if $tJ \geqslant k - 1$ then:

$$E\left[\prod_{i \in I} X_i\right] \geqslant \frac{(tJ)^{\underline{k}}}{t^{\underline{k}}} \ .$$

Proof: Recall the definition of the hash functions $h_i = (b_i, v_i)$. Define $\mathcal{T} = (T_0, T_1, \dots, T_{2t-1})$ in the following way. Let $T_0 = b_0(A \cup B)$ and for $i \geqslant 1$ let $T_i = b_i(A \cup B) \setminus (T_0 \cup \dots \cup T_{i-1})$. Assume in the following that $\mathcal{T}$ is fixed. It clearly suffices to prove this theorem for all possible choices of $\mathcal{T}$. Let $n = |A \cup B|$, then $nJ = |A \cap B|$.

We will prove the claim when the set I is chosen uniformly at random among the subsets of $[t]$ of size k. Because of symmetry this will suffice. More specifically let $I = \{z_0, z_1, \dots, z_{k-1}\}$ where z_i is chosen uniformly at random from $[t] \backslash \{z_0, z_1, \dots, z_{i-1}\}$.

Let $i \in [k]$. Fix $z_0, z_1, \dots, z_{i-1}$ and assume that $X_{z_0} = \dots = X_{vzi-1} = 1$. Let p be the probability that $X_{z_i} = 1$ conditioned on these assumptions. We will estimate p. Let $I' = \{z_0, \dots, z_{i-1}\}$. The probability that $z_i \in T_j$ is then $\frac{|T_j| - |T_j \cap I'|}{t-i}$. Conditioned on $z_i \in T_j$ the probability that $X_{z_i} = 1$ is exactly $\frac{nJ - |T_j \cap I'|}{n - |T_j \cap I'|}$. So the probability that $X_{z_i} = 1$ is:

$$p = \sum_{j \in [2t]} \frac{|T_j| - |T_j \cap I'|}{t-i} \cdot \frac{nJ - |T_j \cap I'|}{n - |T_j \cap I'|} .$$

We note that

$$J \geqslant \frac{nJ - |T_j \cap I'|}{n - |T_j \cap I'|} \geqslant \frac{|T_j| J - |T_j \cap I'|}{|T_j| - |T_j \cap I'|} ,$$

and inserting these estimates gives that:

$$\begin{aligned} J &= \sum_{j \in [2t]} \frac{|T_j| - |T_j \cap I'|}{t-i} \cdot J \\ &\geqslant p \\ &\geqslant \sum_{j \in [2t]} \frac{|T_j| - |T_j \cap I'|}{t-i} \cdot \frac{|T_j| J - |T_j \cap I'|}{|T_j| - |T_j \cap I'|} \\ &= \frac{tJ - i}{t - i} . \end{aligned}$$

So conditioned on $X_{z_0} = \dots = X_{z_{i-1}} = 1$ we conclude that the probability that $X_{z_i} = 1$ is between J and $\frac{tJ-i}{t-i}$. This implies that that the expected value of $\prod_{i \in [k]} X_{z_i}$ is at most J^k and at least $\frac{(tJ)^{\underline{k}}}{t^{\underline{k}}}$ where the lower bound holds if all terms in the product are non-negative, i.e. if $tJ \geqslant k - 1$. ∎

As a corollary we immediately get that the estimator is unbiased.

Lemma 3. *Let A, B be sets with Jaccard similarity $J(A,B) = J$ and let t be a positive integer. Let $X_i = [S(A,t)[i] = S(B,t)[i]]$ and let $X = \sum_{i \in [t]} X_i$. Then $E[X] = tJ$.*

Proof: This follows directly by applying Lemma 2 with $k = 1$. ∎

We also get Chernoff-style concentration bounds as follows.

Lemma 4. *Let A, B be sets with Jaccard similarity $J(A,B) = J$ and let t be a positive integer. Let $X_i = [S(A,t)[i] = S(B,t)[i]]$ and let $X = \sum_{i \in [t]} X_i$. Then for $\delta > 0$*

$$\Pr[X \geqslant J(1+\delta)] \leqslant \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}} \right)^t ,$$

$$\Pr[X \leqslant J(1-\delta)] \leqslant \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}} \right)^t .$$

Proof: The upper bound follows from Lemma 2 and [16, Corollary 1] since Chernoff bounds are derived by bounding $\mathrm{E}\left[e^{\lambda X}\right]$ for some $\lambda > 0$.

The lower bounds follows from considering $Y = \sum_{i \in [t]} Y_i$ where $Y_i = 1 - X_i$ and $Y = t - X$. Since $Y_i = [S(A \cup B, t)[i] = S((A \cup B) \backslash (A \cap B), t)[i]]$ we can use the same argument as for the upper bound, see [16, Page 4]. ∎

Practical implementation: In Figure 2 we used $2t$ hash functions to implement our new similarity sketch. We now briefly describe how to avoid this requirement by instead using just *one* Mixed Tabulation hash function as introduced by Dahlgaard et al. [8]. We do not present the entire details, but refer instead to the theorems of [8] which can be used directly in a black-box fashion.

In tabulation-based hashing we view each key, $x \in [u]$, as a vector $(x_0, \dots, x_{c-1})$ of c characters, where each $x_i \in [u^{1/c}]$, and $u^{1/c}$ is called the *alphabet size*. Consider now the following change to Figure 2: Let h be a mixed tabulation function with alphabet size at least $\delta \cdot t \log n$ for some sufficiently large constant δ, and change Line 5 to be $b, v \leftarrow h(i, a_0, \dots, a_{c-1})$ instead. We now consider two cases:

- If $|A| \leqslant (\delta - 1) \cdot t \log n$ it follows from [8, Theorem 1], that the keys of $\{0, \dots, i\} \times A$ all hash independently, where i is an integer chosen similarly as in Lemma 1, and both correctness and running time follows immediately from the lemmas above.
- If $|A| > (\delta - 1) \cdot t \log n$ all bins are filled out using $i = 0$. In this case both correctness and running time follows immediately from [8, Theorem 2].

A. Separation

It can be useful to check if the Jaccard similarity of two sets are above a certain threshold or not, without having to actually calculate the Jaccard similarity. Specifically, we assume that we are given two sets A and B and want to determine if $J(A,B) \geqslant \gamma$. Intuitively, this should be easy if $J(A,B)$ is either much larger or much smaller than γ and difficult when $J(A,B) \approx \gamma$. Inspired by Henzinger and Thorup [10] we consider the following algorithm for doing so: We let $t \geqslant r$ be positive integers and let $X_i = [S(A,t)[i] = S(B,t)[i]]$ for $i \in [t]$. We now run a for loop with an index i going from r to t. At each step we check if $\sum_{j<i} X_j \leqslant i \cdot \gamma + \sqrt[3]{i^2}$. If so the algorithm terminates and returns `false`. If no such i is found the algorithm returns `true`. See Figure 3 for pseudo-code.

Assume that we use Figure 3 with $X_i = [S(A,t)[i] = S(B,t)[i]]$. In Lemma 5 we show how the algorithm behaves when $J(A,B) \geqslant \gamma + \delta$ and $J(A,B) \leqslant \gamma - \delta$ respectively. Furthermore, if we only count the running time of the algorithm in case the algorithm returns `false` the expected used time is $O(r)$. If $J(A,B) \geqslant \gamma + \delta$, δ is a constant and r is a sufficiently large constant (depending on δ) then the algorithm returns `true` with constant probability.

```
Input: $t, (X_0, X_1, \dots, X_{t-1}), r, \gamma$
Output: true or false
1: $S = 0$
2: for $i = 1, 2, \dots, t$ do
3:     $S = S + X_{i-1}$
4:     if $i \geqslant r$ and $S \leqslant i \cdot \gamma + \sqrt[3]{i^2}$ then
5:         return false
6:     end if
7: end for
8: return true
```

Figure 3. The Separate procedure.

If $J(A, B) \leqslant \gamma - \delta$ and δ is a constant then the algorithm returns true with probability exponentially small in t.

Lemma 5. *Let $t \geqslant r$ be integers and $\gamma, \delta, p \in [0, 1]$. Let $X_0, X_1, \dots, X_{t-1}$ be independent variables with values in $[0, 1]$ such that $E[X_i] = p$ for every $i \in [t]$. Assume that we run Figure 3 with parameters $(t, (X_0, \dots, X_{t-1}), \gamma, r)$.*

Let τ be the number of iterations of the for loop during the algorithm, and let $\tau_F = \tau$ if the algorithm returns false and let $\tau_F = 0$ otherwise. Then $E[\tau_F] = O(r)$.

If $p \geqslant \gamma + \delta$ and $r \geqslant \frac{8}{\delta^3}$ the algorithm returns true with probability at least

$$1 - \frac{e^{-\delta^2 r/2}}{1 - e^{-\delta^2/2}} \,. \tag{2}$$

If $p \leqslant \gamma - \delta$ the algorithm returns true with probability at most

$$e^{-2\delta^2 t} \,. \tag{3}$$

Proof: See Section A. ■

B. Streaming

We note that the algorithm defined in Figure 2 does not work in a streaming context, as it makes several passes over the entire set. This is done to efficiently find the *maximum hash index*. Recall, that the maximum hash index is the smallest index, i, such that each bin is assigned an element when restricting to $h_0, \dots, h_i$. For instance, when $n = \Omega(t \log t)$ we expect the maximum hash index to be $O(1)$. In a streaming setting, if we know what the maximum hash index, i, is, we may simply construct our sketch by computing $h_0(a), \dots, h_i(a)$ once a new element, a, is read from the stream, updating the sketch accordingly. However, it is impossible to know what the maximum hash index is or even estimate it if we don't know the elements that will appear in the stream or even the number of elements. Thus one may fear that we need to evaluate each of the $2t$ hash functions for each element appearing in the stream giving a total time of $O(nt)$ – no better than MinHash. Below we describe how to alleviate this problem by finding the maximum hash index for a prefix of the stream adaptively.

Let $a_1, \dots, a_n$ be the elements in the stream and let $i(a_j)$ be the maximum hash index for $a_1, \dots, a_j$. Clearly, if we have the sketch for $a_1, \dots, a_{j-1}$ we can compute the sketch for $a_1, \dots, a_j$ using only $h_0, \dots, h_{i(a_j)}$. Furthermore, once we read a_j from the stream and compute $h_0, \dots, h_{i(a_j)}$ we can easily detect that $i(a_j)$ is the maximum hash index of $a_1, \dots, a_j$ – e.g. by keeping a counter of bins whose smallest value was assigned using each h_i and updating this after processing each element of the stream.

To analyze this proposed method, we first note that $\mathrm{E}[i(a_j)] \leqslant \left\lceil \frac{t \log t}{j} \right\rceil$, and thus the expected running time of updating the sketch after reading each element is bounded by

$$\begin{aligned} \sum_{j=1}^{n} \mathrm{E}[i(a_j)] &\leqslant \sum_{j=1}^{n} \left\lceil \frac{t \log t}{j} \right\rceil \\ &\leqslant t \log t \log n + n \\ &= O(t \log^2 t + n) \,. \end{aligned}$$

We may augment this simple streaming algorithm with a buffer of size b, where we read b elements at a time and update the current sketch using the algorithm of Figure 2 on the elements in the buffer. By doing this we get an expected total running time of $O(t \log t \log \frac{n}{b} + n)$. As an example, if we allow $O(t)$ space for the buffer (as for the sketch), this gives a total running time of $O(t \log t \log \log t + n)$.

III. Speeding up LSH

We consider the approximate similarity search problem with parameters $0 < j_2 < j_1 < 1$ on a collection, $\mathcal{C}$, of n sets from a large universe $[u]$. We will create a data-structure similar to the LSH structure as described in Section I-A with parameters L and K. That is, for each set $A \in \mathcal{C}$ (and query Q) we will create L sketches $S_0(A), \dots, S_{L-1}(A)$ of size K such that for any two sets $A, B \subseteq [u]$ and $i \in [L]$ we have the following property:

- $\Pr[S_i(A) = S_i(B)] \leqslant J(A, B)^K$.
- If $J(A, B) \geqslant j_1$ then $\Pr[S_i(A) = S_i(B)] = \Theta(J(A, B)^K)$.

By setting $K = \left\lceil \frac{\log n}{\log(1/j_2)} \right\rceil$ and $L = \lceil (1/j_1)^K \rceil$ and using the analysis of [11] this immediately gives us $O(n^{1+\rho} + \sum_{A \in \mathcal{C}} |A|)$ space usage and $O(n^\rho \log n + T(n^\rho, \log n, |Q|))$ expected query time, where $T(L, K, z)$ is the time it takes to create L sketches of size K for a set of size z. By providing a more efficient way to compute the sketches $S_i(A)$ we thus obtain a faster query time.

In order to create the sketches $S_0(A), \dots, S_{L-1}(A)$ described above we first create a $L \times K$ table T such that for each $i \in [L]$ and $j \in [K]$ we have $T[i, j]$ is a uniformly random integer chosen from $\{j \cdot t/K, \dots, (j+1) \cdot t/K - 1\}$, where t is a parameter divisible by K to be chosen later. The rows of the matrix are chosen independently. Each row is filled using a 2-independent source of randomness. Now for a given $A \in [u]$ (or Q) we do as follows:

1) Let $S(A)$ be a size t similarity sketch of Section II.
2) For each $i \in [L]$ and $j \in [K]$ let $S_i(A)[j]$ be $S(A)[T[i,j]]$.

It follows that the time needed to create $S_0(A), \dots, S_{L-1}(A)$ for any $A \in [u]$ is $O(LK + t\log t + |A|)$. We let $t = K \cdot \left\lceil 1 + K \cdot \left(\frac{1}{j_1} - 1\right)\right\rceil$.

We start by bounding the number of "false positives".

Lemma 6. *Let $A \in \mathcal{C}$ be such that $J(A,Q) \leqslant j_2$. Then for any $i \in [L]$ the probability that $S_i(A) = S_i(Q)$ is at most $\frac{1}{n}$.*

Proof: Fix $T[i,j]$ and let $v_j = T[i,j]$ for all $j \in [K]$. Now define $(X_j)_{j \in [t]}$ as in Lemma 2. Then $S_i(A) = S_i(Q)$ if and only if $X_{v_j} = 1$ for all $j \in [K]$, i.e. if $\prod_{j \in [K]} X_{v_j} = 1$. By Lemma 2 this happens with probability at most $(J(A,Q))^K \leqslant j_2^K \leqslant 1/n$. ■

Lemma 6 shows that for each $i \in [L]$ the expected number of sets $A \in \mathcal{C}$ with $J(A,Q) \leqslant j_2$ and $S_i(A) = S_i(Q)$ is at most $|\mathcal{C}| \cdot \frac{1}{n} = 1$. Thus, the expected number of pairs $(i,A) \in [L] \times \mathcal{C}$ with $J(A,Q) \leqslant j_2$ and $S_i(A) = S_i(Q)$ is at most L.

Let $A_0 \in \mathcal{C}$ be a set such that $J = J(A_0,Q) \geqslant j_1$. We will give a lower bound on the probability that there exists an index $i \in [L]$ such that $S_i(A_0) = S_i(Q)$. For $i \in [L]$ let $Y_i = [S_i(A_0) = S_i(Q)]$ and let $Y = \sum_{i \in [L]} Y_i$. Using Lemma 2 and the same reasoning as in Lemma 6 we see that $\mathrm{E}[Y_i] \geqslant \frac{(Jt)^{\underline{K}}}{t^{\underline{K}}}$. Using this we get:

$$\begin{aligned}\mathrm{E}[Y_i] &\geqslant \frac{(Jt)^{\underline{K}}}{t^{\underline{K}}} \\ &> \left(\frac{tJ - K}{t - K}\right)^K \\ &= J^K \cdot \left(1 - \frac{K(1-J)}{J(t-K)}\right)^K .\end{aligned}$$

By the definition of t we have that $\frac{K(1-J)}{J(t-K)} \leqslant \frac{1}{K}$. Hence $\mathrm{E}[Y_i] \geqslant J^K \cdot \left(1 - \frac{1}{K}\right)^K \geqslant J^K/4$. As a consequence we get that $\mathrm{E}[Y] \geqslant L \cdot J^K/4 \geqslant L \cdot j_1^K/4 \geqslant \frac{1}{4}$, i.e. that the expected number of indices $i \in [L]$ such that $S_i(A_0) = S_i(Q)$ is $\Omega(1)$. However, this does not suffice that such an index exists with constant probability. In order to prove this we will bound $\mathrm{E}\left[Y^2\right]$ and use the inequality $\Pr[Y > 0] \geqslant \frac{(\mathrm{E}[Y])^2}{\mathrm{E}[Y^2]}$, which follows from Cauchy-Schwarz's Inequality.

Lemma 7. *Let $i_0, i_1 \in [L]$ be different indices. Then*

$$E[Y_{i_0} Y_{i_1}] \leqslant J^{2K} \cdot \left(1 + \frac{K(1-J)}{Jt}\right)^K .$$

Proof: The values $(T[i,j])_{(i,j) \in \{i_0,i_1\} \times [L]}$ are all independent by definition. Let R be the set containing these value, i.e.

$$R = \{T[i,j] \mid (i,j) \in \{i_0, i_1\} \times [L]\} .$$

Define $X_j = [S(A_0)[j] = S(Q)[j]]$ as in Lemma 2 and fix the value of R. Then by Lemma 2 $\mathrm{E}[Y_{i_0} Y_{i_1} \mid R] \leqslant J^{|R|}$. It remains to understand $|R|$. For $j \in [K]$ let $Z_j = [T[i_0,j] \neq T[i_1,j]]$. Then it is easy to see that $|R| = K + \sum_{j \in [K]} Z_j$, that $(Z_j)_{j \in [K]}$ are independent and that $\Pr[Z_j = 1] = 1 - \frac{K}{t}$. Hence we can upper bound $\mathrm{E}[Y_{i_0} Y_{i_1}]$ by

$$\mathrm{E}[Y_{i_0} Y_{i_1}] \leqslant \mathrm{E}\left[J^K \prod_{j \in [K]} J^{Z_j}\right] = J^K \prod_{j \in [K]} \mathrm{E}\left[J^{Z_j}\right] .$$

Now

$$\mathrm{E}\left[J^{Z_j}\right] = \left(1 - \frac{K}{t}\right) \cdot J + \frac{K}{t} = J \cdot \left(1 + \frac{K(1-J)}{Jt}\right) ,$$

and therefore $\mathrm{E}[Y_{i_0} Y_{i_1}] \leqslant J^{2K} \cdot \left(1 + \frac{K(1-J)}{Jt}\right)^K$. ■

By the definition of t we have $1 + \frac{K(1-J)}{Jt} \leqslant 1 + \frac{1}{K} \leqslant e^{1/K}$, and therefore Lemma 7 gives that $\mathrm{E}[Y_{i_0} Y_{i_1}] \leqslant eJ^{2K}$. Hence:

$$\begin{aligned}\mathrm{E}[Y(Y-1)] &= \sum_{i_0, i_1 \in L, i_0 \neq i_1} \mathrm{E}[Y_{i_0} Y_{i_1}] \\ &\leqslant L(L-1) \cdot e \cdot J^{2K} \\ &< L^2 \cdot e \cdot J^{2K} .\end{aligned}$$

Since $\mathrm{E}[Y] \leqslant L \cdot J^K$ we get that $\mathrm{E}\left[Y^2\right] \leqslant L \cdot J^K + L^2 \cdot e \cdot J^{2K}$. So the probability that $Y > 0$ can be bounded below as follows:

$$\begin{aligned}\Pr[Y > 0] &\geqslant \frac{(\mathrm{E}[Y])^2}{\mathrm{E}[Y^2]} \\ &\geqslant \frac{(L \cdot J^K)^2/16}{e(L \cdot J^K)^2 + (L \cdot J^K)} \\ &= \frac{1}{16(e + (L \cdot J^K)^{-1})} \\ &\geqslant \frac{1}{16(e+1)} \\ &= \Omega(1) .\end{aligned}$$

Avoiding false positives: We let $M = \{(i,A) \in [L] \times \mathcal{C} \mid S_i(A) = S_i(Q)\}$ be the set of matches. We have proved that for each $A_0 \in \mathcal{C}$ with $J(A_0,Q) \geqslant j_1$ with probability $\Omega(1)$ there exists $i \in [L]$ such that $(i, A_0) \in M$. Furthermore, we have proved that the expected number of pairs $(i,A) \in M$ with $J(A,Q) \leqslant j_2$ is at most L. Naively, we could go through all the elements in M until we find $(i,A) \in M$ such that $J(A,Q) > j_2$ in $O(|Q|)$ time per pair. The expected time would be $O(L \cdot |Q|)$, since in expectation we would check $\leqslant L$ pairs (i,A) with $J(Q,A) \leqslant j_2$.

In order to obtain a expected running time of $O(L \cdot |Q|)$ we do something different. We split it into two cases depending on whether $|M| \geqslant CL$ or $|M| \leqslant CL$ for some sufficiently large constant C depending on j_1, j_2. We can in $O(L)$ time check if $|M| \geqslant CL$. First assume that $|M| \geqslant CL$. Then we find a subset $M' \subseteq M$ of size $|M'| = \lceil CL \rceil$, which we can clearly do in $O(L)$ time. Then we sample a uniformly random pair $(i, A) \in M'$ and check if $J(A, Q) > j_2$. By Markov's inequality the number of pairs $(i, A) \in M$ with $J(A, Q) \leqslant j_2$ is at most $\frac{CL}{2}$ with probability $\geqslant 1 - \frac{2}{C}$, and in this case we find a set A with $J(A, Q) \geqslant j_2$ with probability at least $\frac{1}{2}$. The time used in this case is clearly $O(L + |Q|)$.

Now assume that $|M| \leqslant CL$. We assume that we have made a similarity sketch of size $\Theta(\log n)$ for each set $A \in C$ and Q - the running time and space usage for this is clearly dominated by what is used for the sketch of size t. For each $(i, A) \in M$ we now use Figure 3 with $\gamma = \frac{j_1 + j_2}{2}$ on this sketch to separate $J(A, Q)$. We choose r to be a sufficiently large constant. If the algorithm returns `true` we calculate $J(A, Q)$ and if it returns `false` we discard A. We note that for any set A with $J(A, Q) \leqslant j_2$ the probability that we the algorithm returns `true` is at most $\frac{1}{n}$. Hence the expected number of sets $A \in \mathcal{C}$ with $J(A, Q) \leqslant j_2$ for which we calculate $J(A, Q)$ explicitly is at most $O(1)$. We conclude that the running time is $O(L + |Q|)$, since if we calculate $J(A, Q)$ for a set A with $J(A, Q) > j_2$ we can terminate the algorithm and return A. Furthermore, if there exists a set $A_0 \in \mathcal{C}$ with $J(A_0, Q) \geqslant j_1$ the probability that the algorithm returns `true` is $\Omega(1)$ since r is sufficiently large and so there is probability $\Omega(1)$ of finding a set with Jaccard similarity $> j_2$ in this case.

If there exists a set A_0 with Jaccard similarity $J(A_0, Q) \geqslant j_1$ we conclude that the probability of finding a set A with $J(A, Q) > j_2$ is therefore at least $\Omega(1) - \frac{2}{C}$. By choosing C sufficiently large we ensure that $\Omega(1) - \frac{2}{C} = \Omega(1)$.

Summarizing we get the following theorem.

Theorem 2. *Let $0 < j_2 < j_1 < 1$ be constants, and let $\rho = \frac{\log(1/j_1)}{\log(1/j_2)}$. Let U be a set of elements and let $\mathcal{C}$ be collection of n sets from U. Then there exists a data structure using space $O\left(n^{1+\rho} + \sum_{A \in \mathcal{C}} |A|\right)$ and has query time $O(n^\rho \log n + |Q|)$ such that: Given a set Q if there exists a set $A_0 \in \mathcal{C}$ with $J(A_0, Q) \geqslant j_1$, then with constant probability the data structure returns a set $A \in \mathcal{C}$ with $J(A_0, Q) > j_2$.*

In order to improve the probability of Theorem 2 from $\Omega(1)$ to $1 - 1/n$ we can repeat the experiment $\log n$ times. Naively this would require $\log n$ different sketches of size $\Theta(\log^2 n)$, but in the full version we show that it suffices with a single sketch of size $\Theta(\log^3 n)$.

Acknowledgment

The research of Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Mikkel Thorup is partly supported by Mikkel Thorup's Advanced Grant DFF-0602-02499B from the Danish Council for Independent Research under the Sapere Aude research career programme.

The research of Mathias Bæk Tejs Knudsen is partly supported by the FNU project AlgoDisc - Discrete Mathematics, Algorithms, and Data Structures.

References

[1] A. Andoni and I. P. Razenshteyn, "Optimal data-dependent hashing for approximate near neighbors," in *Proc. 47th ACM Symposium on Theory of Computing (STOC)*, 2015, pp. 793–801.

[2] A. Andoni, I. P. Razenshteyn, and N. S. Nosatzki, "LSH forest: Practical algorithms made theoretical," in *Proc. 28th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, 2017, pp. 67–78.

[3] Y. Bachrach and E. Porat, "Sketching for big data recommender systems using fast pseudo-random fingerprints," in *Proc. 40th International Colloquium on Automata, Languages and Programming (ICALP)*, 2013, pp. 459–471.

[4] R. J. Bayardo, Y. Ma, and R. Srikant, "Scaling up all pairs similarity search," in *Proc. 16th WWW*, 2007, pp. 131–140.

[5] A. Z. Broder, "On the resemblance and containment of documents," in *Proc. Compression and Complexity of Sequences (SEQUENCES)*, 1997, pp. 21–29.

[6] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, "Syntactic clustering of the web," *Computer Networks*, vol. 29, pp. 1157–1166, 1997.

[7] T. Christiani and R. Pagh, "Set similarity search beyond minhash," *CoRR*, vol. abs/1612.07710, 2016, to appear at STOC'17.

[8] S. Dahlgaard, M. B. T. Knudsen, E. Rotenberg, and M. Thorup, "Hashing for statistics over k-partitions," in *Proc. 56th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2015, pp. 1292–1310.

[9] M. R. Henzinger, "Finding near-duplicate web pages: a large-scale evaluation of algorithms," in *Proc. 29th ACM SIGIR Conference on Research and Development in Information Retrieval*, 2006, pp. 284–291.

[10] M. R. Henzinger and M. Thorup, "Sampling to provide or to bound: With applications to fully dynamic graph algorithms," *Random Struct. Algorithms*, vol. 11, no. 4, pp. 369–379, 1997.

[11] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. 13th ACM Symposium on Theory of Computing (STOC)*, 1998, pp. 604–613.

[12] P. Li, "0-bit consistent weighted sampling," in *Proc. 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 665–674.

[13] P. Li, A. B. Owen, and C.-H. Zhang, "One permutation hashing," in *Proc. 26th Advances in Neural Information Processing Systems*, 2012, pp. 3122–3130.

[14] P. Li, A. Shrivastava, J. L. Moore, and A. C. König, "Hashing algorithms for large-scale learning," in *Proc. 25th Advances in Neural Information Processing Systems*, 2011, pp. 2672–2680.

[15] M. Pătraşcu and M. Thorup, "On the k-independence required by linear probing and minwise independence," in *Proc. 37th International Colloquium on Automata, Languages and Programming (ICALP)*, 2010, pp. 715–726.

[16] J. P. Schmidt, A. Siegel, and A. Srinivasan, "Chernoff-hoeffding bounds for applications with limited independence," *SIAM Journal on Discrete Mathematics*, vol. 8, no. 2, pp. 223–250, 1995.

[17] G. Shakhnarovich, T. Darrell, and P. Indyk, "Nearest-neighbor methods in learning and vision," *IEEE Trans. Neural Networks*, vol. 19, no. 2, p. 377, 2008.

[18] A. Shrivastava, "Optimal densification for fast and accurate minwise hashing," in *Proc. 34th International Conference on Machine Learning (ICML)*, 2017, pp. 3154–3163.

[19] A. Shrivastava and P. Li, "Densifying one permutation hashing via rotation for fast near neighbor search," in *Proc. 31th International Conference on Machine Learning (ICML)*, 2014, pp. 557–565.

[20] ——, "Improved densification of one permutation hashing," in *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence, UAI 2014, Quebec City, Quebec, Canada, July 23-27, 2014*, 2014, pp. 732–741.

[21] M. Thorup, "Bottom-k and priority sampling, set similarity and subset sums with minimal independence," in *Proc. 45th ACM Symposium on Theory of Computing (STOC)*, 2013.

[22] S. Tong, "Lessons learned developing a practical large scale machine learning system," April 2010. [Online]. Available: https://research.googleblog.com/2010/04/lessons-learned-developing-practical.html

[23] K. Q. Weinberger, A. Dasgupta, J. Langford, A. J. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *Proc. 26th International Conference on Machine Learning (ICML)*, 2009, pp. 1113–1120.

Appendix

Proof: For $i = 0, 1, 2, \ldots, n$ we let $X_{<i} = \sum_{j<i} X_j$.

First we prove the bound on the expected value of τ_F. For $j = r, r+1, \ldots, t$ let $T_j = j$ if the algorithm returns `false` when $i = j$ in the loop and let $T_j = 0$ otherwise. We clearly have that $\tau_F = r + \sum_{i=r}^{t} T_i$. Clearly $\sum_{i=r}^{2r} T_i \leqslant 2r$ by definition, and therefore

$$\tau_F \leqslant 3r + \sum_{i=2r+1}^{t} T_i\,.$$

Now fix $i > 2r$. If $T_i = i$ then we must have that $X_{<i} \geqslant i \cdot \gamma + \sqrt[3]{i^2}$. Let $j = \lfloor i/2 \rfloor$. Since the algorithm did not stop earlier we must also have $X_{<j} < j \cdot \gamma + \sqrt[3]{j^2}$. Hence we have that:

$$\mathrm{E}[T_i] \leqslant i \cdot \min \begin{cases} \Pr\Big[X_{<i} \geqslant i \cdot \gamma + \sqrt[3]{i^2}\Big], \\ \Pr\Big[X_{<j} < j \cdot \gamma + \sqrt[3]{j^2}\Big] \end{cases}.$$

Let $\gamma' = \gamma + \frac{1}{2} \cdot \left(\frac{1}{\sqrt[3]{i}} + \frac{1}{\sqrt[3]{j}}\right)$. If $p \leqslant \gamma'$ then we see that if $X_{<i} \geqslant i \cdot \gamma + \sqrt[3]{i^2}$ then:

$$X_{<i} - \mathrm{E}[X_{<i}] \geqslant i \cdot \gamma + \sqrt[3]{i^2} - i\gamma' = \Omega\left(\sqrt[3]{i^2}\right).$$

And by Hoeffding's inequality we conclude that:

$$\Pr\Big[X_{<i} - \mathrm{E}[X_{<i}] \geqslant i \cdot \gamma + \sqrt[3]{i^2} - i\gamma'\Big] \leqslant e^{-\Omega(\sqrt[3]{i})}.$$

If $p > \gamma'$ we conclude in the same manner that $\Pr\Big[X_{<j} < j \cdot \gamma + \sqrt[3]{j^2}\Big] \leqslant e^{-\Omega(\sqrt[3]{i})}$. Hence we get that:

$$\mathrm{E}\left[\sum_{i=2r}^{t} T_i\right] \leqslant \sum_{i=2r}^{t} i \cdot e^{-\Omega(\sqrt[3]{i})} \leqslant \sum_{i \geqslant 1} i \cdot e^{-\Omega(\sqrt[3]{i})} = O(1)\,.$$

We conclude that $\mathrm{E}[\tau_F] = O(r)$ as desired.

We now assume that $p \geqslant \gamma + \delta$ and prove that (2) is a lower bound on the probability that `true` is returned. By a union bound and Hoeffding's inequality we get that the probability that `false` is returned is at most

$$\begin{aligned} &\sum_{i=r}^{t} \Pr\Big[X_{<i} \leqslant i \cdot \gamma + \sqrt[3]{i^2}\Big] \\ &\quad \leqslant \sum_{i=r}^{t} \Pr[X_{<i} \leqslant i \cdot (\gamma + \delta/2)] \\ &\quad \leqslant \sum_{i=r}^{t} \Pr[X_{<i} - \mathrm{E}[X_{<i}] \leqslant -i \cdot \delta/2] \\ &\quad \leqslant \sum_{i=r}^{t} e^{-\delta^2 i/2} \\ &\quad \leqslant \sum_{i \geqslant r} e^{-\delta^2 i} = \frac{e^{-\delta^2 r/2}}{1 - e^{-\delta^2/2}}\,, \end{aligned}$$

as desired.

Now assume that $p \geqslant \gamma - \delta$. We note that `true` is only returned if $X_{<t} > t \cdot \gamma$. By Hoeffding's inequality this happens with probability at most

$$\begin{aligned} \Pr\Big[X_{<t} > t \cdot \gamma + \sqrt[3]{t^2}\Big] &\leqslant \Pr[X_{<t} > t \cdot \gamma] \\ &\leqslant \Pr[X_{<t} - \mathrm{E}[X_{<t}] > t \cdot \delta] \\ &\leqslant e^{-2\delta^2 t}\,, \end{aligned}$$

showing that (3) holds. ■