Rejection Sampling for Weighted Jaccard Similarity Revisited

Xiaoyun Li, Ping Li
Cognitive Computing Lab
Baidu Research
10900 NE 8th St. Bellevue, WA 98004, USA
{lixiaoyun996, pingli98}@gmail.com

Abstract

Efficiently[1] computing the weighted Jaccard similarity has become an active research topic in machine learning and theory. For sparse data, the standard technique is based on the *consistent weighed sampling (CWS)*. For dense data, however, methods based on rejection sampling (RS) can be much more efficient. Nevertheless, existing RS methods are still slow for practical purposes. In this paper, we propose to improve RS by a strategy, which we call *efficient rejection sampling (ERS)*, based on "early stopping + densification". We analyze the statistical property of ERS and provide experimental results to compare ERS with RS and other algorithms for hashing weighted Jaccard. The results demonstrate that ERS significantly improves the existing methods for estimating the weighted Jaccard similarity in relatively dense data.

Introduction

The Jaccard is a similarity measure in machine learning such as classification tasks (Li 2017, 2018). Given two non-negative data vectors $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}_+^D$, the Jaccard is defined as

$$J(\boldsymbol{v}, \boldsymbol{w}) = \frac{\sum_{i=1}^D \min\{v_i, w_i\}}{\sum_{i=1}^D \max\{v_i, w_i\}}. \tag{1}$$

To deal with data with negative entries, Li (2017, 2018) proposed the "generalized min-max (GMM)" kernel. See the experiments which considered data with negative entries.

For binary $(0/1)$ data, (1) is also known as the resemblance, which is an important metric in numerous applications. The standard hashing algorithm for computing the resemblance is based minwise hashing (Broder et al. 1997, 1998; Li and König 2011). In the past two or three decades, the resemblance and minwise hashing have found numerous practical applications (Fetterly et al. 2003; Jindal and Liu 2008; Buehrer and Chellapilla 2008; Urvoy et al. 2008; Dourisboure, Geraci, and Pellegrini 2009; Forman, Eshghi, and Suermondt 2009; Pandey et al. 2009; Cherkasova et al. 2009; Chierichetti et al. 2009; Gollapudi and Sharma 2009; Najork, Gollapudi, and Panigrahy 2009; Bendersky and Croft 2009; Li et al. 2011; Shrivastava and Li 2012; Schubert, Weiler, and Kriegel 2014; Fu et al. 2015; Pewny et al. 2015; Manzoor, Milajerdi, and Akoglu 2016; Raff and Nicholas 2017; Zhu et al. 2019; Lei et al. 2020).

For general non-binary data, the Jaccard similarity and efficient computational methods have been extensively studied (Kleinberg and Tardos 1999; Charikar 2002; Gollapudi and Panigrahy 2006; Chierichetti et al. 2010; Hadjieleftheriou and Srivastava 2010; Ioffe 2010; Manasse, McSherry, and Talwar 2010; Bollegala, Matsuo, and Ishizuka 2011; Delgado et al. 2014; Wang et al. 2014; Li 2015; Shrivastava 2016; Ertl 2018; Bag, Kumar, and Tiwari 2019; Li, Li, and Zhang 2019; Pouget-Abadie et al. 2019; Yang et al. 2019; Fuchs et al. 2020; Li et al. 2021). Generally speaking, there are two categories of hashing methods which are commonly used for efficiently computing the Jaccard similarity.

The first category is the *consistent weighted sampling (CWS)* (Manasse, McSherry, and Talwar 2010; Ioffe 2010; Li 2015), which is suitable for sparse data. To generate K hash samples, CWS has $\mathcal{O}(Kd)$ complexity where d is the number of non-zero elements in the vector. Thus, CWS is efficient when data are very sparse. Some recent work (e.g., BagMinHash (Ertl 2018), DartMinHash (Christiani 2020)) improves CWS in some cases, but the complexity is still increasing with d. The work on "0-bit CWS" (Li 2015) provided a practical (and heuristic) simplification of CWS and the more recent work (Li et al. 2021) developed the rigorous theory from the perspective of extremal processes.

The second category of methods for hashing weighted Jaccard similarity is the *rejection sampling (RS)* (Kleinberg and Tardos 1999; Charikar 2002; Shrivastava 2016). In particular, the algorithm in Shrivastava (2016) involves repeatedly sampling random sequences until some criteria is satisfied (more details will be introduced later on). The complexity of RS relies on a totally different quantity named *effective sparsity*. In some sense, RS hits the opposite side of CWS, since it runs very fast on dense data, but possibly very slowly on sparse data. Moreover, the time complexity of RS is unbounded in the worst case, which may lead to a huge computational cost. In this paper, we revisit the RS algorithm and propose a variant that makes RS much more efficient for practical applications.

Rejection Sampling (RS) for Weighted Jaccard

Setting. Suppose we have a non-negative data vector $\boldsymbol{v} \in \mathbb{R}_+^D$, and we know the upper bound of the data entries for every dimension. Let us denote these upper limits by $m_i, i =$

[1] *We sincerely thank the constructive comments from Reviewers. The work was initially submitted in February 2020 to ICML'20.*

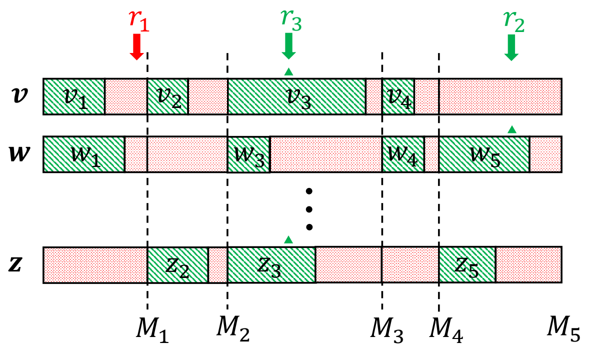

Figure 1: Illustration of RS algorithm: $\boldsymbol{v} = (v_1, v_2, v_3, v_4, 0)$ $\boldsymbol{w} = (w_1, 0, w_3, w_4, w_5)$ and $\boldsymbol{z} = (0, z_2, z_3, 0, z_5)$. The hash samples are $h(\boldsymbol{v}) = 3$, $h(\boldsymbol{w}) = 2$ and $h(\boldsymbol{z}) = 3$.

$1, ..., D$. Denote the cumulative sum $M_i = \sum_{d=1}^{i} m_d$. As demonstrated in Figure 1, we split the area into two regions. The green region represents the parts covered by the data values, and the red region otherwise. More specifically, the two regions are defined as

$$\text{Green} = \bigcup_{i=1}^{d} [M_{i-1}, M_{i-1}+v_i], \;\; \text{Red} = \bigcup_{i=1}^{d} [M_{i-1}+v_i, M_i].$$

Algorithm. The RS algorithm (Shrivastava 2016) is summarized in Algorithm 1. The function ISGREEN(r,$\boldsymbol{v}$) checks if a number r falls into the green region formulated by vector $\boldsymbol{v}$, which could be reduced to constant time complexity by constructing an extra hash map. To generate one hash sample, we generate a uniform random number on $[0, M_D]$ and check if it belongs to the green region. If not, we continue with the another random number. The hash sample, $h(\boldsymbol{v})$, is simply the number of attempts until a random jump hits the green region. An example is provided in Figure 1. By fundamental probabilistic argument, we can show that

$$P[h(\boldsymbol{v}) = h(\boldsymbol{w})] = J(\boldsymbol{v}, \boldsymbol{w}). \tag{2}$$

Algorithm 1 Rejection Sampling (RS) for Weighted Jaccard

```
Input: data $\boldsymbol{v} \in \mathbb{R}_+^D$, $M_D = \sum_{d=1}^{D} m_d$, random $seed[\,]$
Initialize: hash samples $h[\,]$
For $k = 1$ to $K$
    $randseed = seed[k]$
    While true
        $h[k] = h[k] + 1$
        $r = M_D \times Uniform[0, 1]$
        If ISGREEN($r, \boldsymbol{v}$)
            break;
        End If
        $randseed = \lceil r \times 10^6 \rceil$
    End While
End For
Output: $K$ hash samples $h[\,]$
```

Thus, an unbiased estimator of $J(\boldsymbol{v}, \boldsymbol{w})$ can be constructed by independently repeating the procedure for K times,

$$\hat{J}_{RS}(\boldsymbol{v}, \boldsymbol{w}) = \frac{1}{K} \sum_{k=1}^{K} \mathbf{1}\{h_k(\boldsymbol{v}) = h_k(\boldsymbol{w})\}. \tag{3}$$

Time Complexity. The running time of RS algorithm essentially depends on the following quantity.

Definition 1. *The effective sparsity of a data vector $\boldsymbol{v}$ is defined as*

$$s_v = \frac{S(Green)}{S(Green) + S(Red)} = \frac{\sum_{i=1}^{D} v_i}{M_D}, \tag{4}$$

where $S(\cdot)$ denotes the size of red or green region.

In this paper, "sparsity" will refer to this definition. It is easy to show that $\mathbb{E}[h(\boldsymbol{v})] = 1/s_v$, i.e., the computational cost decreases as s_v increases. In applications where the effective sparsity s_v is small, we will need a large number of jumps before stopping. Therefore, the rejection sampling scheme may run slowly on sparse datasets. On the other hand, for relatively dense data, RS can be very efficient.

From Random Seeds to Tabulation

The key to ensure the unbiasedness of $\hat{J}_{RS}$ is the consistency of the random numbers, meaning that for all data vectors we should use the same set of random r_i's. In hashing algorithms such as CWS, the common practical implementation to achieve consistency is to pre-generate (tabulate) all the random numbers a priori, and read from the fixed tables throughout the hashing process, which is called *tabulation*. In general, as table look-up is faster than generating seeded random numbers iteratively, tabulation is a useful trick for practical speedup. For example, CWS with tabulation consistently runs faster than standard implementation (Christiani 2020). In Algorithm 1, the consistency is ensured by repeatedly sampling random numbers based on (pseudo-fixed) random seeds on the fly, until one number hits the green region. Compared to tabulation, this strategy is slow.

There are challenges if we hope to apply tabulation ideas in RS. Consider tabulating down a $K \times L$ random matrix with uniform $[0, M_D]$ entries, from which we use the i-th row (a fixed length-L sequence) to generate the i-th hash sample. One critical problem we confront is that, for any data vector $\boldsymbol{v}$ with sparsity $s_v \neq 1$, since $h(\boldsymbol{v})$ follows a geometric distribution which is unbounded, there is always a non-zero probability that the random jump does not fall into green region within L steps. This causes the problem of *invalid hash sample* that we have to deal with. In other words, we have to establish an "early stopping" mechanism for RS. Indeed, since invalid hash samples are always possible to appear whenever we choose to fix the random numbers beforehand (i.e., tabulation), a specific construction of such mechanism is indispensable for rigorous algorithm design, to make the algorithm self-contained. After that, we can safely use fast tabulation trick in rejection sampling.

Our Contributions

- We propose Efficient Rejection Sampling (ERS) based on an "early stopping + densification" scheme to handle the invalid hash samples, which only requires pre-generating fixed-length random sequences.
- We provide the theoretical analysis on the mean and variance of the ERS Jaccard estimator. In particular, under our tabulated ERS framework, the estimator is always slightly biased upwards, though negligibly small in most cases.
- Experiments are conducted to compare the efficiency of different hashing methods, showing that ERS can be significantly faster in common scenarios when data are not extremely sparse. In particular, ERS substantially accelerates the original RS (Algorithm 1). We provide empirical evidence on the proper choice of sequence length L that brings two sources of acceleration (one from early stopping and one from tabulation), and meanwhile maintains good estimation and learning performance.

Efficient Rejection Sampling (ERS) with Tabulation: Early Stop + Densification

As explained earlier, if we wish to read from tabulated tables during the hashing process of RS, invalid hash samples are always possible. This issue will not be fixed even if we tabulate down very long random sequences. To tackle this problem, we propose a scheme called Efficient Rejection Sampling (ERS) based on early stopping + densification.

As shown in Algorithm 2, the idea is simple: we use independent random sequences with fixed-length L for K hash samples, which is pre-generated a priori. Here L can be chosen flexibly. When $L \to \infty$, the ERS approximates the original RS method (because the probability of getting invalid hash samples goes to 0). When L is set relatively small, during the hashing process, if the sequence does not fall into the green region till the end (i.e., within L random jumps), we record an "E" representing an empty hash. After going through K independent sequences, we obtain a vector of hash samples denoted as h', possibly containing empty hashes. As the second step, h' is processed with a densification procedure, where we replace each "E" with a non-empty hash chosen uniformly at random. This algorithm is described in Algorithm 3, which was introduced to densify the bin-wise min-hashing (Shrivastava and Li 2014; Shrivastava 2017; Li, Li, and Zhang 2019). Here, the 2-universal hashing function $\mathcal{H} : [K] \times \mathbb{N} \to [K]$ realizes the goal of choosing a non-empty hash uniformly at random, while avoiding cycles. The second input of $\mathcal{H}(\cdot,\cdot)$ counts the number of attempts made so far to find a non-empty hash.

Algorithm 2 Efficient Rejection Sampling (ERS)

```
Input: data vector $v \in \mathbb{R}_+^D$, $M_D = \sum_{d=1}^D m_d$, random Uniform$[0, M_D]$ matrix $R \in \mathbb{R}^{K\times L}$
Initialize: $h'[\,] = E$
For $k = 1$ to $K$
    For $i = 1$ to $L$
        If ISGREEN($R[k,i], v$)
            $h'[k] = i$
        End If
    End For
End For
$h^*[\,]$ = Densification($h'[\,]$)
Output: Hash vector $h^*[\,]$
```

Algorithm 3 Densification for ERS method

```
Input: hash samples $h'[\,]$, 2-U hashing function $\mathcal{H}(\cdot,\cdot)$
Initialize: $h^*[\,] = 0$
For $k = 1$ to $K$
    If $h'[k] \neq E$
        $h^*[k] = h'[k]$
    Else
        $attempt = 1$
        $next = \mathcal{H}(k, attempt)$
        While $h'[next] = E$
            $attempt = attempt + 1$
            $next = \mathcal{H}(k, attempt)$
        End While
        $h^*[k] = h'[next]$
    End If
End For
Output: The densified hash vector $h^*[\,]$
```

Time Complexity. The theoretical expected complexity to generate K hash samples of ERS for vector $\boldsymbol{v}$ is

$$\mathcal{O}\left(K\left(\min(L, \frac{1}{s_v}) + \frac{N_{emp}}{K - N_{emp}}\right)\right),$$

where N_{emp} is the number of empty hashes. Here, the second term is usually small if we choose L properly such that there is a sufficient number of non-empty hashes. Thus, the worst complexity is in general upper bounded by $\mathcal{O}(KL)$. As a comparison, recall that the running time of the original RS is $\mathcal{O}(K/s_v)$ and $\mathcal{O}(Kd)$ for CWS where d is the number of non-zero elements in $\boldsymbol{v}$. As we can see, when L is set to be smaller than $1/s_v$ and d, ERS is theoretically more efficient than the original RS and CWS. Importantly, we clarify that this acceleration comes from reduced theoretical complexity brought by the "early stopping" strategy. Since tabulation provides additional acceleration in terms of practical implementation, we consequently expect a significant acceleration of ERS over RS in practice.

Theoretical Analysis

Suppose $\boldsymbol{v}$ and $\boldsymbol{w}$ are two non-negative data vectors with effective sparsity s_v and s_w respectively, and the Jaccard similarity equals to J. Without loss of generality, we scale everything by $1/M_D$. Then the area of intersection is precisely

$$\delta = \frac{J}{1+J}(s_v + s_w). \tag{5}$$

To estimate the Jaccard similarity using our proposed scheme, we define the ERS estimator

$$\hat{J}_{ERS}(\boldsymbol{v}, \boldsymbol{w}) = \frac{1}{K}\sum_{k=1}^{K} \mathbf{1}\{h_k^*(\boldsymbol{v}) = h_k^*(\boldsymbol{w})\}, \tag{6}$$

where h^* is the output after densification (Algorithm 3).

Theorem 1. *For data vector $\boldsymbol{v}$, denote m as the expected number of non-empty hash samples in ERS (Algorithm 2). Then L should satisfy $L \geq \frac{\log(1-m/K)}{\log(1-s_v)}$.*

In Theorem 1, m/K is the fraction of non-empty hash samples before densification, which should not be too small (otherwise there is too little information). We can show that if a data vector has sparsity $s_v = 10^{-p}$, then $L = \frac{1}{s_v} = 10^p$ would make $m/K \approx 0.65$, and $L = \frac{1}{2s_v}$ gives $m/K \approx 0.4$.

Expectation

We can classify a pair of hash samples $(h(\boldsymbol{v}), h(\boldsymbol{w}))$ into four categories:

- Type 0: $(\blacklozenge, \blacklozenge)$, both hash samples are non-empty.
- Type $1v$: $(E, \blacklozenge)$, only $h(\boldsymbol{v}) = E$; Type $1w$: $(\blacklozenge, E)$, only $h(\boldsymbol{w}) = E$.
- Type 2: (E, E), both hash samples are empty.

The next lemma gives some useful probabilistic results.

Lemma 1. *Let p_0, p_1 and p_2 be the probability of one pair of hash samples being type 0, $1v$, $1w$ and 2, respectively. Let $\delta = \frac{J}{1+J}(s_v + s_w)$, $p_v = 1 - (1 - s_v)^L$ and $p_w = 1 - (1 - s_w)^L$. We have*

$$p_{1v} = (1 - p_v)\left[1 - \left(\frac{1 - s_v - s_w + \delta}{1 - s_v}\right)^L\right],$$

$$p_{1w} = (1 - p_w)\left[1 - \left(\frac{1 - s_v - s_w + \delta}{1 - s_w}\right)^L\right],$$

$$p_2 = (1 - s_v - s_w + \delta)^L, \qquad p_0 = 1 - p_{1v} - p_{1w} - p_2.$$

In Shrivastava (2016), it was shown that the original RS estimator $\hat{J}_{RS}$ is unbiased. However, this is under the assumption that the length of random sequence can be infinite ($L = \infty$ as Algorithm 1). As we will show next, with our early stopping strategy, the estimator $\hat{J}_{ERS}$ becomes biased due to the possible accidental collision in the densification process. The precise result is provided as follows.

Theorem 2. *Suppose $\boldsymbol{v}$ and $\boldsymbol{w}$ are two non-negative vectors with Jaccard J and effective sparsity s_v and s_w, respectively. Then,*

$$\mathbb{E}[\hat{J}_{ERS}] = J + \frac{p_1}{p_v p_w}\frac{s_v s_w}{s_v + s_w - s_v s_w}, \tag{7}$$

where p_v, p_w and $p_1 = p_{1v} + p_{1w}$ are given by Lemma 1.

Remark 1. *When $L \to \infty$ (the original RS algorithm), we have $p_1 \to 0$, $p_v, p_w \to 1$ and thus $\mathbb{E}[\hat{J}_{ERS}] \to J$.*

Theorem 2 says that when tabulation is used, the ERS estimator is always biased upwards. This is a crucial difference between ERS (Algorithm 2) and the original RS (Algorithm 1). The theoretical biases plotted in Figure 2 show:

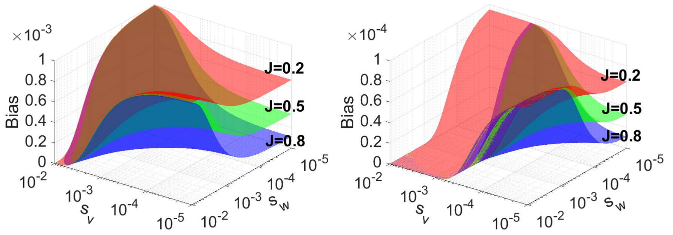

Figure 2: Bias of $\hat{J}_{ERS}$ with different s_v, s_w and J. Left panel: $L = 1000$. Right panel: $L = 10000$.

- The bias decreases as L or J increases.
- Even for very sparse vectors, e.g., $s_v = s_w = 10^{-5}$ such that $1/s = 10^5$, the bias is already negligibly small (less than 10^{-3}) with L merely equals to 1000.

The important message of above analysis is that, when L is chosen not too small (depending on different sparsity level), the bias of ERS estimator is usually negligible.

Mean Squared Error (MSE)

The estimation accuracy of an estimator $\hat{J}$ is can be measured by Mean Squared Error (MSE), which is defined as

$$MSE(\hat{J}) = \mathbb{E}[(\hat{J} - J)^2].$$

In the sequel, we study the MSE of $\hat{J}_{ERS}$. Note that for completeness, we consider a more flexible setting where we may choose to use the first $K' \leq K$ hash samples for estimation, although in total K hash samples are generated.

Theorem 3. *Under the same setting and notations as Lemma 1 and Theorem 2, denote $\mu = \mathbb{E}[\hat{J}_{ERS}]$ and let $p = 1 - (1 - s_v - s_w + \delta)^L$. The Mean Squared Error (MSE) of $\hat{J}_{ERS}$ is given by*

$$MSE(\hat{J}_{ERS}) = \frac{\mu}{K'} - 2\mu J + J^2 +$$
$$\sum_{j=0}^{K'}\left[j(j-1)\tilde{E}_0 + (K'-j)(K'+j-1)\tilde{E}_1\right]\frac{B(K', j, p)}{K'^2},$$

where $B(K', j, p) = \binom{K'}{j} p^j (1-p)^{K'-j}$ is the binomial density. Let p_{1v} and p_{1w} be defined in Lemma (1). We have

$$\begin{aligned}\tilde{E}_0 = \sum_{m=1}^{K}\Big[J^2 &+ \frac{p_{1v}^2\Theta_{1v,1v}(m)}{(p_0+p_1)^2} + \frac{p_{1w}^2\Theta_{1w,1w}(m)}{(p_0+p_1)^2} \\ &+ \frac{2p_0p_{1v}\Theta_{0,1v}(m)}{(p_0+p_1)^2} + \frac{2p_0p_{1w}\Theta_{0,1w}(m)}{(p_0+p_1)^2} \\ &+ \frac{2p_{1v}p_{1w}\Theta_{1v,1w}(m)}{(p_0+p_1)^2}\Big]B(K, m, p_0+p_1) \\ \triangleq \sum_{m=1}^{K} V_{df}(m)&B(K, m, p_0+p_1),\end{aligned}$$

$$\tilde{E}_1 = \sum_{m=1}^{K} \Big[\frac{1}{m}V_{sm}(m) + \frac{m-1}{m}V_{df}(m)\Big] B(K, m, p_0 + p_1),$$

where

$$V_{sm}(m) = J + \frac{p_{1v}\Theta_{1v}}{p_0 + p_1} + \frac{p_{1w}\Theta_{1w}}{p_0 + p_1}.$$

Let $\tilde{p}_{0v} = p_0 + p_{1v}$, $\tilde{p}_{0w} = p_0 + p_{1w}$, $\tilde{p} = p_0 + p_1$. *We have*

$$\Theta_{1v,1v}(m) = \sum_{j=1}^{m-2} \Big[\frac{G_{1v2w}}{j} + \frac{j-1}{j}\triangle^2\Big] B(m-2, j, \tilde{p}_{0w}),$$

$$\Theta_{1w,1w}(m) = \sum_{j=1}^{m-2} \Big[\frac{G_{1w2v}}{j} + \frac{j-1}{j}\triangle^2\Big] B(m-2, j, \tilde{p}_{0v}),$$

$$\Theta_{0,1v}(m) = \sum_{j=0}^{m-2} \Big[\frac{G_{0,1v}}{j+1} + \frac{j}{j+1}\frac{\tilde{p}}{p_0}J\triangle\Big] B(m-2, j, \tilde{p}_{0w}),$$

$$\Theta_{0,1w}(m) = \sum_{j=0}^{m-2} \Big[\frac{G_{0,1w}}{j+1} + \frac{j}{j+1}\frac{\tilde{p}}{p_0}J\triangle\Big] B(m-2, j, \tilde{p}_{0v}),$$

$$\Theta_{1v,1w}(m) = \sum_{\substack{x+y+z=m-2\\ x,y,z\geq 0}} p_0^x p_{1v}^y p_{1w}^z \frac{(m-2)!}{x!y!z!}\Big\{\frac{\triangle}{n_v n_w} + \frac{n_v - 1}{n_v n_w}G_{1w2v} + \frac{n_w - 1}{n_v n_w}G_{1v2w} + [1 - \frac{n_v + n_w - 1}{n_v n_w}]\triangle^2\Big\},$$

$$\Theta_{1v} = \sum_{j=1}^{m-1} \Big[\frac{1}{j}\triangle + \frac{j-1}{j}G_{1w2v}\Big] B(m-1, j, \tilde{p}_{0w}),$$

$$\Theta_{1w} = \sum_{j=1}^{m-1} \Big[\frac{1}{j}\triangle + \frac{j-1}{j}G_{1v2w}\Big] B(m-1, j, \tilde{p}_{0v}),$$

where $n_v = x + z + 1$ *and* $n_w = x + y + 1$. *In addition,*

$$\triangle = \frac{s_v s_w}{p_v p_w}\frac{1 - (1 - s_v - s_w + s_v s_w)^L}{s_v + s_w - s_v s_w},$$

$$G_{1v2w} = \frac{s_v s_w^2}{p_v p_w^2}\frac{1 - (1 - s_v)^L (1 - s_w)^{2L}}{1 - (1 - s_v)(1 - s_w)^2},$$

$$G_{1w2v} = \frac{s_v^2 s_w}{p_v^2 p_w}\frac{1 - (1 - s_v)^{2L} (1 - s_w)^{L}}{1 - (1 - s_v)^2(1 - s_w)},$$

$$G_{0,1v} = \frac{\delta s_w}{p_0 p_w}\frac{1 - (1 - s_v - s_w + \delta)^L (1 - s_w)^L}{1 - (1 - s_v - s_w + \delta)(1 - s_w)},$$

$$G_{0,1w} = \frac{\delta s_w v}{p_0 p_v}\frac{1 - (1 - s_v - s_w + \delta)^L (1 - s_v)^L}{1 - (1 - s_v - s_w + \delta)(1 - s_v)}.$$

Remark 2. *As* $L \to \infty$, $p_0 \to 1$ *and* $p_1, p_2 \to 0$. *We can verify that* $MSE(\hat{J}_{ERS}) \to \frac{J(1-J)}{K'} = MSE(\hat{J}_{RS})$.

The theoretical bias and MSE of ERS will be empirically validated in next section (see Figure 3).

Experiments

In this section, we test the efficiency and estimation quality of ERS. In addition, we also show that it works well in linearized kernel learning, a potential application of ERS.

Running Time Comparison

There have been several hashing algorithms proposed for weighted Jaccard similarity in recent years. However, a comprehensive comparison of efficiency is still missing in literature, particularly for rejection sampling methods. In the following, we compare the running time of ERS with various hashing methods, to confirm its significant efficiency advantage when data are not extremely sparse. All the tests are run using C++ on an Intel(R) Xeon(R) Platinum 8276 CPU 2.20GHz server with optimization flag -O3 enabled. Unless specifically stated, the pseudo-random number generator (PRNG) is the Mersenne Twister (mt19937) generator. For competing methods CWS, BagMinHash and DartMinHash, we use the same source code as in Christiani (2020). Specifically, the codes are optimized to have fast practical speed. Interestingly, tabulation is also used in these implementations for acceleration.

CWS. We implement an optimized version. In particular, the random numbers are tabulated down and the use of logarithm, exponential and division is minimized. The logarithm of data $\boldsymbol{v}$ is computed only once for all K hash samples.

BagMinHash. The implementation is from the original source code of Ertl (2018). We choose to use version "BagMinHash2" since it is faster. The PRNG is XXHash64.

DartMinHash. We use the implementation provided by Christiani (2020). Some pre-computed tables are used to make the code faster. Also, the code uses fast tabulation hashing (Zobrist 1990) to simulate random draws.

RS. For the original RS algorithm, we strictly follow the architecture in Algorithm 1, where random numbers are successively produced until a random jump hits the green region. In implementation, we do not need to reset the seed for every single random number inside the WHILE since the mt19937 generator naturally produces the same sequence of random numbers once the initial seed is the same. We observe that it indeed accelerates the algorithm by 1$\sim$2x.

ERS. We implement ERS (Algorithm 2) with various L. It is convenient to measure L in terms of $1/s$ (recall s is the effective sparsity). We test $L = \alpha\frac{1}{s}$ with $\alpha = 0.5, 1, 5$.

Following e.g., Ertl (2018), we will use synthetic data to flexibly control the data statistics (e.g., norm and sparsity). The dimensionality is set to $D = 2^{16} = 65,536$, and we vary the number of non-zeros d. Each non-zero entry is i.i.d. standard uniform. Notably, DartMH is fastest when $\|\boldsymbol{v}\|_1 = 1$. Thus, for DartMH we normalize the data vector to have unit l_1 norm to favor DartMH. We randomly generate 100 data vectors and report the average time for generating $K = \{256, 512, 1024\}$ hash samples. Table 1 shows that:

- ERS is fastest when $d \geq 2^{10}$, and significantly so as d gets larger. It can be 10$\sim$40x faster than the best competitor DartMH, and 100$\sim$1000x faster than CWS. Note that $d = 2^{10}$ corresponds to $d/D \approx 1.5\%$, which is already fairly sparse in terms of non-zeros.
- ERS with $\alpha = 5$ is substantially faster than vanilla RS ($\sim$5x), which demonstrates the *practical speedup* of us-

d	K	CWS	BagMH	DartMH	RS	ERS $\alpha = 0.5$	$\alpha = 1$	$\alpha = 5$
512	256	2.61	10.44	0.44	6.24	**0.36**	1.06	1.38
512	512	4.97	19.69	**0.68**	12.57	1.16	1.96	3.11
512	1024	10.11	42.27	**1.26**	26.65	2.59	4.27	6.48
1024	256	4.61	11.23	0.58	2.83	**0.16**	0.34	0.58
1024	512	9.44	22.42	0.86	6.61	**0.39**	0.90	1.48
1024	1024	19.14	45.69	1.48	13.76	**1.34**	2.26	3.49
2048	256	5.76	12.20	0.70	2.44	**0.10**	0.15	0.48
2048	512	12.89	22.83	1.02	5.34	**0.19**	0.39	1.26
2048	1024	27.74	47.12	1.84	10.61	**0.61**	1.30	2.59
4096	256	18.05	18.39	0.90	0.80	**0.06**	0.08	0.14
4096	512	35.77	30.17	1.54	1.59	**0.11**	0.20	0.28
4096	1024	71.26	55.82	2.27	3.74	**0.22**	0.43	0.88
8192	256	22.12	19.84	1.60	0.71	**0.04**	0.06	0.11
8192	512	49.44	34.40	2.01	1.39	**0.08**	0.11	0.25
8192	1024	105.51	61.87	3.03	3.08	**0.15**	0.22	0.61
16384	256	70.93	33.65	2.79	0.46	**0.03**	0.04	0.04
16384	512	140.02	52.68	3.29	0.79	**0.05**	0.08	0.10
16384	1024	278.05	89.01	4.13	1.73	**0.10**	0.14	0.23

Table 1: Running time (ms) of different algorithms. The results are averaged over 100 random and independent repetitions.

ing tabulation (since $\alpha = 5$ is large enough to mimic Algorithm 1 with tabulation only). Also, ERS is more efficient with smaller α. This acceleration comes from *reduced complexity* brought by early stopping in ERS.

Inevitably, by the nature of rejection sampling technique, ERS will become slower in sparser data. Nevertheless, as demonstrated by our results, ERS is the most efficient method as long as the data are not extremely sparse. For relatively dense data, ERS can be faster by magnitudes.

Estimation and Learning

We validate the results in Theorem 2 and Theorem 3, and provide experiments on two applications: Jaccard estimation and kernel learning. We use the *Words* dataset (Li and Church 2005) and *Caltech101* (Li, Fergus, and Perona 2007) (which was also used in Shrivastava (2016)). The *Words* dataset contains 2702 samples, and each instance is a word count in 2^{16} different documents. The average number of non-zeros is 2.11×10^3 and the average reciprocal of sparsity (i.e., the expected number of jumps) is 2.04×10^3. This dataset is in general fairly sparse. For *Caltech101*, we use the gray scale pixel values as the data samples, randomly selected from 11 classes of images for Jaccard estimation. On average, $D = 485,640$, $d = 66,390$ and $1/s \approx 17$.

Jaccard Estimation. In Figure 3, we validate our theoretical formulas from Theorem 2 and Theorem 3 with empirical mean and MSE on word pair AGAIN-AGAINST. The Jaccard similarity is $J = 0.147$, while the sparsity s is 4.6×10^{-3}, 4.1×10^{-3} for two word vectors, respectively. The number of non-zeros d is 3,655 and 2,945, respectively. We see that the empirical value overlaps the theory.

To test the estimation accuracy, we perform evaluation over whole datasets. In Figure 4, we draw the mean absolute error (MAE) of ERS Jaccard estimator, over all sample-pairs in *Words* and *Caltech101* datasets, along with the time ratio of ERS and RS over CWS. We see that with $L = 10^3$ (i.e., $\alpha \approx 0.5$), the error curve of ERS almost overlaps that of CWS for *Words*, and $L = 50$ suffices for *Caltech101*. On both datasets, with properly chosen L, ERS 1) improves RS by a large factor (10∼50x), and 2) beats CWS in efficiency (∼3x for *Words* and more than 1000x for *Caltech101*), while providing the same estimation accuracy. Note that, a very small L would greatly increase the chance of accidental collision in densification of ERS, leading to poor estimates. Consequently, when $1/s$ is very small, we recommend using a larger L, e.g., $\alpha \geq 50/17 \approx 3$ to ensure estimation quality.

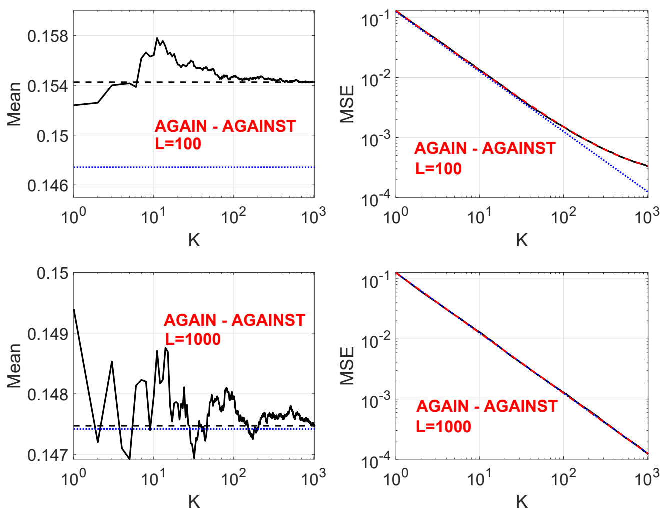

Figure 3: Mean and MSE of ERS estimator of a word pair. Blue dash curves are true J, and the black dotted curves are the theoretical mean of ERS. Black solid curves are empirical mean and MSE, respectively.

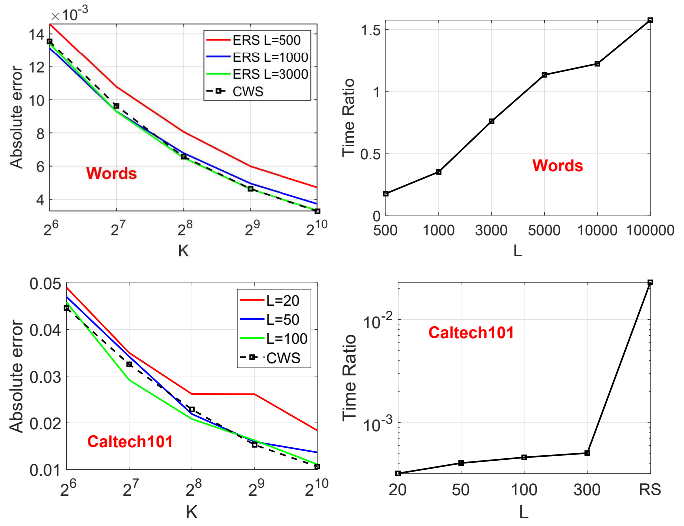

Figure 4: *Words* and *Caltech101*: MAE over all sample pairs, and running time ratio of ERS and RS over CWS.

Classification. Li (2015, 2017); Li and Zhang (2017); Li (2018) showed that using the Jaccard similarity as the kernel often leads to much better classification accuracy than using linear kernel. One can directly and (very efficiently) use hash samples from CWS, RS, or ERS as linear kernel to approximate the classification result from Jaccard kernel. As Li (2015, 2017); Li and Zhang (2017); Li (2018) already provided very exhaustive empirical results, here we only use two datasets from the UCI repository[2] as shown in Table 2 to confirm the advantage of ERS. Both datasets are randomly split 50/50 for training and testing. Data features are pre-processed to have unit l_2 norm.

	n	D	C	d	$1/s$
Dailysports	9120	11250	19	5590	294.6
PCMAC	1932	3289	2	47.9	628.2

Table 2: Statistics of datasets. C is the number of classes, d is the average number of non-zeros and $1/s$ is the average reciprocal of effective sparsity. Since *Dailysports* contains negative entries, we apply the (GMM) transformation (Li 2017) to generate a new dataset which doubles the original dimension and contains only non-negative entries.

Figure 5 reports the test accuracy of CWS and ERS with different choice of L. On both datasets, we see that the learning performance of ERS elevates as L increases. For *Dailysports* ($1/s \approx 300$), L =100∼300 gives accuracy very close to that of exact GMM kernel. On *PCMAC* ($1/s \approx 650$), $L = 300$ suffices. That said, setting $L \approx \frac{1}{2s}$ (i.e., $\alpha \approx 0.5$) is sufficient for matching the GMM baseline in approximate kernel learning. Nevertheless, one may also choose to use a more conservative (slightly larger) L in practice for more robustness. This would make ERS slightly slower than using a small L, but still much faster than RS. Again, one may flexibly choose L to find a good balance.

[2]http://archive.ics.uci.edu/ml

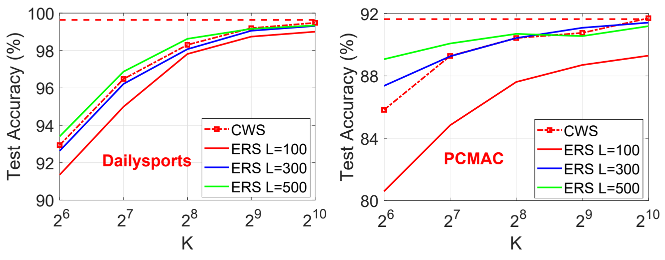

Figure 5: Test accuracy of SVM classification, for comparing ERS ($L = \{100, 300, 500\}$) with CWS. The dash line denotes the accuracy from using the original kernel.

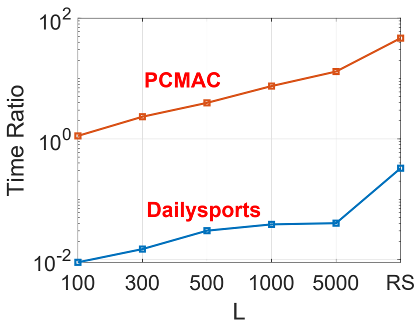

Figure 6: The running time ratio of ERS and RS over CWS, on two classification datasets.

In Figure 6, we observe similar trend of time ratio for these datasets. On *Dailysports* which is relatively dense, ERS with $L = 300$ is substantially faster than CWS and RS. On *PCMAC* which is very sparse, note that CWS would run $628/48 \approx 13$ times faster than RS in principle. While with $L = 300$, ERS only requires double running time as CWS. This illustrates that on sparse data where CWS has a natural speed advantage over RS, our fast ERS can be comparable to CWS in terms of efficiency.

Conclusion

In this present paper, we mainly address the efficiency issue of using rejection sampling (RS) in the hashing process of weighted Jaccard similarity. We propose our ERS strategy based on the "early stopping + densification" scheme, which allows the use of fast tabulation trick to accelerate the original RS algorithm in terms of both the practical implementation time and the theoretical complexity. We show how the random sequence length L controls the efficiency-accuracy trade-off in ERS, and a proper L in practice can provide both fast speed and good estimation quality. Moreover, we also provide comprehensive time comparison among various hashing algorithms, confirming the superior efficiency of ERS in the case when data are not extremely sparse.

References

Bag, S.; Kumar, S. K.; and Tiwari, M. K. 2019. An efficient recommendation generation using relevant Jaccard similarity. *Information Sciences* 483: 53–64.

Bendersky, M.; and Croft, W. B. 2009. Finding text reuse on the web. In *Proceedings of the Second International Conference on Web Search and Web Data Mining (WSDM)*, 262–271. Barcelona, Spain.

Bollegala, D.; Matsuo, Y.; and Ishizuka, M. 2011. A Web Search Engine-Based Approach to Measure Semantic Similarity between Words. *IEEE Trans. Knowl. Data Eng.* 23(7): 977–990.

Broder, A. Z.; Charikar, M.; Frieze, A. M.; and Mitzenmacher, M. 1998. Min-Wise Independent Permutations. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing (STOC)*, 327–336. Dallas, TX.

Broder, A. Z.; Glassman, S. C.; Manasse, M. S.; and Zweig, G. 1997. Syntactic clustering of the Web. In *WWW*, 1157 – 1166. Santa Clara, CA.

Buehrer, G.; and Chellapilla, K. 2008. A scalable pattern mining approach to web graph compression with communities. In *Proceedings of the International Conference on Web Search and Web Data Mining (WSDM)*, 95–106. Stanford, CA.

Charikar, M. S. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing (STOC)*, 380–388. Montreal, Canada.

Cherkasova, L.; Eshghi, K.; III, C. B. M.; Tucek, J.; and Veitch, A. C. 2009. Applying syntactic similarity algorithms for enterprise information management. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 1087–1096. Paris, France.

Chierichetti, F.; Kumar, R.; Lattanzi, S.; Mitzenmacher, M.; Panconesi, A.; and Raghavan, P. 2009. On compressing social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 219–228. Paris, France.

Chierichetti, F.; Kumar, R.; Pandey, S.; and Vassilvitskii, S. 2010. Finding the Jaccard Median. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 293–311. Austin, TX.

Christiani, T. 2020. DartMinHash: Fast Sketching for Weighted Sets. *arXiv preprint arXiv:2005.11547* .

Delgado, A. D.; Martínez-Unanue, R.; Fresno-Fernández, V.; and Montalvo, S. 2014. A Data Driven Approach for Person Name Disambiguation in Web Search Results. In *Proceedings of the 25th International Conference on Computational Linguistics (COLING)*, 301–310. Dublin, Ireland.

Dourisboure, Y.; Geraci, F.; and Pellegrini, M. 2009. Extraction and classification of dense implicit communities in the Web graph. *ACM Trans. Web* 3(2): 1–36.

Ertl, O. 2018. BagMinHash - Minwise Hashing Algorithm for Weighted Sets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, 1368–1377. London, UK.

Fetterly, D.; Manasse, M.; Najork, M.; and Wiener, J. L. 2003. A large-scale study of the evolution of web pages. In *Proceedings of the Twelfth International World Wide Web Conference (WWW)*, 669–678. Budapest, Hungary.

Forman, G.; Eshghi, K.; and Suermondt, J. 2009. Efficient detection of large-scale redundancy in enterprise file systems. *SIGOPS Oper. Syst. Rev.* 43(1): 84–91. ISSN 0163-5980.

Fu, M.; Feng, D.; Hua, Y.; He, X.; Chen, Z.; Xia, W.; Zhang, Y.; and Tan, Y. 2015. Design Tradeoffs for Data Deduplication Performance in Backup Workloads. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST)*, 331–344. Santa Clara, CA.

Fuchs, G.; Acriche, Y.; Hasson, I.; and Petrov, P. 2020. Intent-Driven Similarity in E-Commerce Listings. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM)*, 2437–2444. Virtual Event, Ireland.

Gollapudi, S.; and Panigrahy, R. 2006. Exploiting asymmetry in hierarchical topic extraction. In *Proceedings of the 2006 ACM CIKM International Conference on Information and Knowledge Management (CIKM)*, 475–482. Arlington, VA.

Gollapudi, S.; and Sharma, A. 2009. An axiomatic approach for result diversification. In *Proceedings of the 18th International Conference on World Wide Web (WWW)*, 381–390. Madrid, Spain.

Hadjieleftheriou, M.; and Srivastava, D. 2010. Weighted Set-Based String Similarity. *IEEE Data Eng. Bull.* 33(1): 25–36.

Ioffe, S. 2010. Improved Consistent Sampling, Weighted Minhash and L1 Sketching. In *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM)*, 246–255. Sydney, AU.

Jindal, N.; and Liu, B. 2008. Opinion spam and analysis. In *Proceedings of the International Conference on Web Search and Web Data Mining (WSDM)*, 219–230. Palo Alto, CA.

Kleinberg, J.; and Tardos, E. 1999. Approximation Algorithms for Classification Problems with Pairwise Relationships: Metric Labeling and Markov Random Fields. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS)*, 14–23. New York, NY.

Lei, Y.; Huang, Q.; Kankanhalli, M. S.; and Tung, A. K. H. 2020. Locality-Sensitive Hashing Scheme based on Longest Circular Co-Substring. In *Proceedings of the 2020 International Conference on Management of Data (SIGMOD)*, 2589–2599. Online conference [Portland, OR, USA].

Li, F.; Fergus, R.; and Perona, P. 2007. Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories. *Comput. Vis. Image Underst.* 106(1): 59–70.

Li, P. 2015. 0-Bit Consistent Weighted Sampling. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 665–674. Sydney, Australia.

Li, P. 2017. Linearized GMM Kernels and Normalized Random Fourier Features. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 315–324.

Li, P. 2018. Several Tunable GMM Kernels. *arXiv preprint arXiv:1805.02830* .

Li, P.; and Church, K. W. 2005. Using Sketches to Estimate Associations. In *Proceedings of the 2005 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 708–715. Vancouver, Canada.

Li, P.; and König, A. C. 2011. Theory and applications of *b*-bit minwise hashing. *Commun. ACM* 54(8): 101–109.

Li, P.; Li, X.; Samorodnitsky, G.; and Zhao, W. 2021. Consistent Sampling Through Extremal Process. In *Proceedings of the Web Conference (WWW)*. Virtual.

Li, P.; Li, X.; and Zhang, C.-H. 2019. Re-randomized Densification for One Permutation Hashing and Bin-wise Consistent Weighted Sampling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 15900–15910. Vancouver, Canada.

Li, P.; Shrivastava, A.; Moore, J.; and König, A. C. 2011. Hashing Algorithms for Large-Scale Learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2672–2680. Granada, Spain.

Li, P.; and Zhang, C.-H. 2017. Theory of the GMM Kernel. In *Proceedings of the 26th International Conference on World Wide Web (WWW)*, 1053–1062. Perth, Australia.

Manasse, M.; McSherry, F.; and Talwar, K. 2010. Consistent Weighted Sampling. Technical Report MSR-TR-2010-73.

Manzoor, E. A.; Milajerdi, S. M.; and Akoglu, L. 2016. Fast Memory-efficient Anomaly Detection in Streaming Heterogeneous Graphs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 1035–1044. San Francisco, CA.

Najork, M.; Gollapudi, S.; and Panigrahy, R. 2009. Less is more: sampling the neighborhood graph makes SALSA better and faster. In *Proceedings of the Second International Conference on Web Search and Web Data Mining (WSDM)*, 242–251. Barcelona, Spain.

Pandey, S.; Broder, A.; Chierichetti, F.; Josifovski, V.; Kumar, R.; and Vassilvitskii, S. 2009. Nearest-neighbor caching for content-match applications. In *Proceedings of the 18th International Conference on World Wide Web (WWW)*, 441–450. Madrid, Spain.

Pewny, J.; Garmany, B.; Gawlik, R.; Rossow, C.; and Holz, T. 2015. Cross-Architecture Bug Search in Binary Executables. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy (SP)*, 709–724. San Jose, CA.

Pouget-Abadie, J.; Aydin, K.; Schudy, W.; Brodersen, K.; and Mirrokni, V. S. 2019. Variance Reduction in Bipartite Experiments through Correlation Clustering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 13288–13298. Vancouver, Canada.

Raff, E.; and Nicholas, C. K. 2017. An Alternative to NCD for Large Sequences, Lempel-Ziv Jaccard Distance. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 1007–1015. Halifax, Canada.

Schubert, E.; Weiler, M.; and Kriegel, H. 2014. SigniTrend: scalable detection of emerging topics in textual streams by hashed significance thresholds. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 871–880. New York, NY.

Shrivastava, A. 2016. Simple and Efficient Weighted Minwise Hashing. In *Neural Information Processing Systems (NIPS)*, 1498–1506. Barcelona, Spain.

Shrivastava, A. 2017. Optimal Densification for Fast and Accurate Minwise Hashing. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 3154–3163. Sydney, Australia.

Shrivastava, A.; and Li, P. 2012. Fast Near Neighbor Search in High-Dimensional Binary Data. In *Proceedings of European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, 474–489. Bristol, UK.

Shrivastava, A.; and Li, P. 2014. Densifying One Permutation Hashing via Rotation for Fast Near Neighbor Search. In *Proceedings of the 31th International Conference on Machine Learning (ICML)*. Beijing, China.

Urvoy, T.; Chauveau, E.; Filoche, P.; and Lavergne, T. 2008. Tracking Web spam with HTML style similarities. *ACM Trans. Web* 2(1): 1–28.

Wang, J.; Song, Y.; Leung, T.; Rosenberg, C.; Wang, J.; Philbin, J.; Chen, B.; and Wu, Y. 2014. Learning Fine-Grained Image Similarity with Deep Ranking. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1386–1393. Columbus, OH.

Yang, D.; Rosso, P.; Li, B.; and Cudré-Mauroux, P. 2019. NodeSketch: Highly-Efficient Graph Embeddings via Recursive Sketching. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, 1162–1172. Anchorage, AK.

Zhu, E.; Deng, D.; Nargesian, F.; and Miller, R. J. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD)*, 847–864. Amsterdam, The Netherlands.

Zobrist, A. L. 1990. A new hashing method with application for game playing. *ICGA Journal* 13(2): 69–73.