On Densification for Minwise Hashing

Tung Mai Adobe
Anup Rao Adobe
Matt Kapilevich Adobe
Ryan Rossi Adobe
Yasin Abbasi-Yadkori VinAI
Ritwik Sinha Adobe

Abstract

One Permutation Hashing (OPH) is a significantly more efficient alternative to the popular minwise hashing. To produce a sketch of size k, OPH requires just one hash function whereas the classical minwise hashing requires k hash functions. However, OPH does not have the desirable locality sensitive hashing (LSH) property that is important for indexing. [Srivastava and Li 2014] proposed the novel idea of densification to produce LSH sketches from OPH sketches, and gave the first densification routine. In this paper, we give a necessary and sufficient condition for a densification routine to result in LSH sketches when applied to OPH sketches. Furthermore, we give a novel densification routine that for every input, takes $O(k \log k)$ time in expectation and achieves better variance than the previous best bound obtained by [Srivastava 2017]. The running time of the densification routine given in [Srivastava 2017] for worst case inputs is $O(k^2)$ in expectation.

1 INTRODUCTION

Many modern data sets are often large but have features that are sparse and binary. Examples include textual documents [Henzinger, 2006], images [Chum et al., 2008] or video frames [Chum et al., 2007] (represented as bags of words) and web behavioural data, for instance computational advertising [Shi et al., 2018] (where categorical data is converted by using one hot encoding). Binary data can be expressed as sets. A widely used metric for set similarity is Jaccard similarity. Given two sets A and B, Jaccard similarity is defined as

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

Often the modern datasets are huge in both the number of data points and the number of features. Sketching is a popular way to effectively reduce the dimensionality of the data while preserving key statistics. Minhash is a widely used method to reduce dimensionality while preserving Jaccard similarity. This has many applications [Broder, 1997, Broder et al., 1997, Bayardo et al., 2007] including duplicate detection [Henzinger, 2006], nearest neighbor search [Indyk and Motwani, 1998] and large scale learning [Li et al., 2011]. In this paper, we consider a more efficient way to sketch sets to obtain constant sized vectors which approximately preserves Jaccard similarity.

Given a set A, we denote its sketch vector by $s(A) = (s(A)_1, ..., s(A)_k)$. In this paper, we only consider sketches which have the *alignment* or *LSH property*,

$$\Pr(s(A)_i = s(B)_i) = J(A, B),$$

for all i. The alignment property is important [Shrivastava, 2017, Dahlgaard et al., 2017] for large scale machine learning and answering approximate nearest neighbor queries. When this holds, an unbiased estimate of the Jaccard similarity of sets A and B is given by $\frac{1}{k}\sum_i \mathbb{1}(s(A)_i = s(B)_i)$. We will assume throughout that all the sets are subsets of $[n]$ for some big integer n. Minhash [Broder, 1997, Broder et al., 1997] is given by $s(A) = (\min_{a\in A}\pi_1(a), \min_{a\in A}\pi_2(a), ..., \min_{a\in A}\pi_k(a))$, where $\pi_i : [n] \to [n], i = 1, ..., k$ are random permutations. This takes $O(k|A|)$ to sketch a set A. One often needs to take k to be around several thousands to get satisfyingly accurate estimate in practice [D. Ondov et al., 2016]. Given the huge amount of data, evaluating k hash function is computationally very expensive, and it is highly desirable to remove the factor k from the running time.

Bottom-k sketch and one permutation hashing scheme (OPH) [Li et al., 2012] are two alternatives that overcome this bottleneck. Both require just one hash evaluation. Bottom-k sketch has a worst case running time of $O(|A| \ln k)$ time and OPH takes $O(|A|)$ time. But neither has the LSH property that is important for many downstream tasks. OPH produces a k dimensional vector where some of the coordinates can be empty. This will be the case when $|A| = o(k \log k)$. To address this, in a series of papers [Shrivastava and Li, 2014a, Shrivastava and Li, 2014b, Shrivastava, 2017], the authors give various *densification* routines to fill in the empty positions by copying values from the non-empty positions. They also show that the resulting sketch has the LSH property. Since one replicates values from non-empty bins, the copying procedure affects the variance of the Jaccard estimate. The densification routine producing the current best variance is given in [Shrivastava, 2017]. The running time of this procedure for worst case inputs is $O(k^2)$ in expectation (the algorithm is randomized), which is attained when the size of the set is very small compared to k. When this densification routine is used along with OPH, the worst case expected running time to sketch a set A is $O(|A| + k^2)$. We also note that in practice one might want to just store and update the sketches produced by OPH (due to potential space savings for small sets), and run the densification routine only when required by certain downstream tasks. In these cases, the running time of the densification routine of [Shrivastava, 2017] for worst case inputs is $O(k^2)$ in expectation (over the randomness in the algorithm). In this paper, we give a much faster densification routine. Our main result is:

Theorem 1.1. *The densification given in Algorithm 1 takes $O(k \log k)$ expected time, produces sketches with LSH property and achieves better variance than the densification proposed in [Shrivastava, 2017].*

Finally, we would like to note that in [Chen and Shrivastava, 2016] the authors apply the densification idea to Winner Takes All (WTA) sketches and show that the resulting densified sketch has a significantly better performance in image search and classification. One can use the densification routine we propose in this paper without affecting their accuracy gains, but obtaining significant computational gains in the very sparse regime.

1.1 OUR CONTRIBUTIONS

The contribution of this paper is two fold. First, we identify a notion of consistency that characterizes valid densification routines. A valid densification routine is a procedure to fill in the empty coordinates by copying values from non-empty coordinates such that the sketch so produced has LSH property. All the densification routines in [Shrivastava and Li, 2014a, Shrivastava and Li, 2014b, Shrivastava, 2017] satisfy this consistency property and hence leads to unbiased estimators. Second, we give a new densification routine whose variance is better than the current best one given in [Shrivastava, 2017], and runs in expected $O(k \log k)$ time. We note that [Shrivastava, 2017] proves that the variance of the densification routine given there is optimal for a restricted class of algorithms. Our algorithm does not belong to the class considered there and hence, we are able to achieve a slightly better variance and achieve nearly linear runtime as a function of k.

In [Dahlgaard et al., 2017], the authors give an algorithm that computes a sketch of A in expected time $O(|A| + k \log k)$, and achieves superior concentration guarantees than the methods based on one permutation hashing. But the algorithm there needs full access to A at the start and is not designed to be a one pass algorithm. In many practical applications, the set of interest keeps growing. A particular example is computational advertising datasets where new users with various categorical attributes get added every day. We would like to maintain a sketch of the sets, updating it using only the current day's data and not have to look back at data already processed once. OPH is well suited in these situations when the set is updated continuously, and we want to continuously maintain a sketch. OPH is a single pass algorithm, and using the densification routine given in this paper, we can compute a sketch in $O(|A| + k \log k)$ expected time and achieve better variance than previous densification routines.

1.2 PRELIMINARIES AND NOTATIONS

Let U denote the population set which we assume is finite in size. When k is a positive integer, we use U^k to denote the set of all k dimensional vectors whose coordinates are in U. Without loss of generality, we assume $U = [n]$ for a large integer n. Let π a random permutation $\pi : U \to U$. For a set $A \subset U$, we let $\pi(A) = \{\pi(a) \,| a \in A\}$. We will also use the notation $\min(A) = \min_{a \in A} a.$. Let π be a random permutation. A minhash sketch of size one of a set $A \subseteq U$ is given by $\pi(A)$. Then, it is not hard to show that the collision probability is given by

$$\Pr\big(\min(\pi(A)) = \min(\pi(B))\big) = \frac{|A \cap B|}{|A \cup B|} = J(A, B). \tag{1}$$

Let $\mathbb{1}(\cdot)$ be a function that takes value 1 when the argument is true, and zero otherwise. Hence, $\mathbb{1}\big(\min(\pi(A)) = \min(\pi(B))\big)$ is an unbiased estimator of $J(A, B)$. A k-minhash sketch $s : 2^U \to U^k$ is a random function obtained by sampling k random permutations $\pi_1, ..., \pi_k$ and defining $s(A) =$

$(\min(\pi_1(A)), ..., \min(\pi_k(A)))$. We have

$$\tilde{J}(s) = \frac{1}{k} \sum_{i \in [k]} \mathbb{1}\big(s(A)_i = s(B)_i\big) \tag{2}$$

is also an unbiased estimator of $J(A, B)$. In other words, the sketches $s(A), s(B)$ of sets A, B can be used to estimate $J(A, B)$ by doing a pair-wise comparison of the coordinates. For any sketch mentioned in the paper, expectation and variance are with respect to that estimator.

1.2.1 One Permutation Hashing

One permutation hashing (OPH) is an efficient way to generate a sketch of size k by partitioning U into k equal sized bins. Formally, let U_i denote the ith partition of U. Then the ith coordinate of the sketch of a set A is given by:

$$OPH(A)_i = \begin{cases} \min\big(\pi(A) \cap U_i\big), & \text{if } \pi(A) \cap U_i \neq \emptyset \\ e, & \text{otherwise.} \end{cases}$$

Here e is special element denoting the empty item. Let $U_e = U \cup \{e\}$. Then $OPH(A) \in U_e^k$ for any $A \subseteq U$.

For $v \in U_e^k$, denote by E_v the set of empty coordinates/bins and N_v the set of non-empty coordinates/bins of v. In other words, $E_v = \{i : v_i = e\}$ and $N_v = \{i : v_i \neq e\}$.

Consider applying one permutation hashing on $A, B \subseteq U$. We say that a bin (or an index) i is simultaneously empty if the ith coordinates of the sketches of both A and B are empty. To be precise, $E_{A,B} = E_{OPH(A)} \cap E_{OPH(B)}$ is the set of all simultaneously empty bins. Similarly, a bin is simultaneously non-empty if either $OPH(A)_i$ or $OPH(B)_i$ (or both) is non-empty. Formally, $N_{A,B} = [k] \setminus E_{A,B} = N_{OPH(A)} \cup N_{OPH(B)}$.

It is proven in [Li et al., 2012] that for all $i \in N_{A,B}$, the property stated in Equation 1 holds:

$$\Pr\big(OPH(A)_i = OPH(B)_i\big) = J(A, B). \tag{3}$$

However, it does not necessarily hold for $i \in E_{A,B}$. Therefore, even though OPH is computationally efficient, it does not have the desirable LSH property.

1.2.2 Densification

The authors of [Shrivastava and Li, 2014a] presented a novel idea of reassigning the values of non-empty bins to empty bins so that the resulting sketch has the LSH property. This is called a densification routine. In the paper, the process they proposed is fast and simple: every empty bin gets assigned the value of the nearest non-empty bin on the circular right. The densification produces a sketch having LSH property. However, the variance of the corresponding estimator is unnecessarily high.

In [Shrivastava and Li, 2014b], the authors showed that utilizing k random bits could improve the variance, while achieving the same running time. In particular, every empty bin receives a random bit and gets assigned the value of the nearest non-empty bin toward circular right (or left) if the bit is 0 (or 1).

However, there is still an issue with the variance obtained by the densification given in [Shrivastava and Li, 2014b], as pointed out by [Shrivastava, 2017]. Specifically, the variance does not go to 0 as k goes to ∞. Hence, [Shrivastava, 2017] proposed another densification scheme, in which the reassigment process is done using 2-universal hash functions. [Shrivastava, 2017] showed that the scheme improved the variance over [Shrivastava and Li, 2014b], and is optimal if the reassigments of any two empty bins are independent.

1.2.3 Consistent Sampling

Consistent sampling is a technique for sampling a subset of elements from a given set so that samples from similar sets are likely to be similar. Minhash sketch, in particular, is consistent. Consistent sampling has crucial applications in estimation of set similarity [Manasse et al., 2010] and number of distinct items in a data stream. In this paper, we are only interested in sampling one element from a set.

Definition 1.2 (Consistent Sampling of One Element)**.** Consistent sampling is a sampling process that returns an element from a given set and satisfies the following: whenever $x \in S$ is sampled from T and $S \subseteq T$ then x is also sampled from S.

1.3 NEW DENSIFICATION ROUTINE

As noted above, the worst case running time of the densification routine given in [Shrivastava, 2017] is $O(k^2)$, which occurs when number of non-empty bins is small compared to k. The reason is, when that happens, an empty bin would need $O(k)$ expected hash functions to find a non-empty bin, since there are few of them. Moreover, this observation holds at every step of the densification.

To circumvent that, we apply hash functions from non-empty bins to empty ones instead. When the number of empty bins becomes small, the expected time needed to fill a bin is also $O(k)$. However, such bad cases only happen toward the end of the routine as opposed to the densification in [Shrivastava, 2017].

Reversing the direction of hash functions can also reduce the variance. In particular, it prevents two empty bins to copy values from the same non-empty bin, in the same round. Hence, the values copied in the same round are more balancedly distributed.

2 DENSIFICATION CHARACTERIZATION

In this section we give a formal definition for densification and a characterization of densification routines that result in sketches with LSH property when applied to the output of OPH.

2.1 Formalization

We define a densification process as follows:

Definition 2.1 (Densification)**.** A densification is a function $D : U_e^k \to U^k$ such that $\forall v \in U_e^k$:

- $\forall i \in N_v : D(v)_i = v_i$
- $\forall i \in E_v, \exists j \in N_v : D(v)_i = v_j$. We say that $f_v(i) = j$, where f_v is a function from E_v to N_v.

In other words, D is a rule that when applied on a vector $v \in U_e^k$, copies values in non-empty bins to empty bins. Given D and v, f_v is the reassignment function such that $f_v(i)$ is the bin whose value is copied to i, for every $i \in E_v$.

Definition 2.2 (Consistent Densification)**.** A densification process is consistent if $\forall v, i \in E_v$, $f_v(i)$ is chosen from N_v according to a consistent sampling.

The process of reassigning values from non-empty bins to empty-bins can be viewed as a sampling process. In particular, each empty bin samples a bin from the set of non-empty ones to copy from. A consistent densification guarantees that the sampling process is consistent. Note that we use 'sampling' even when the procedure is potentially deterministic, and not just uniformly randomly.

It can be checked that all densifications proposed in [Shrivastava and Li, 2014a, Shrivastava and Li, 2014b, Shrivastava, 2017] are consistent. For example, in [Shrivastava and Li, 2014a] the desification process fills an empty bin by using value from the nearest non-empty bin toward circular right. Consider $u, v \in U_e^k$ such that $N_u \subseteq N_v$, and $i \in E_v$. We claim that the sampling process to fill bin i is consistent. To see this, assume $f_v(i) \in N_u$. Then $f_v(i)$ is the bin in N_v that is nearest to i and towards the circular right. Now, if $f_v(i) \in N_u$, then since $N_u \subseteq N_v$, it must also be the nearest bin in N_u. Hence, $f_u(i) = f_v(i)$ as desired.

2.2 A NECESSARY AND SUFFICIENT CONDITION

In this section we give a necessary and sufficient condition for a densification scheme to produce a sketch that has LSH property when applied to OPH sketch. We show

Theorem 2.3. *A densification D is consistent if and only if for all $1 \leq i \leq k$, and all $A, B \subseteq U$*

$$\Pr\big(D(A)_i = D(B)_i\big) = J(A, B).$$

We first state and prove some lemmas from which the proof of the above theorem will be immediate. For simplicity of notations, we will use throughout the paper $D(A)$ and $D(B)$ to represent $D(OPH(A))$ and $D(OPH(B))$, where A and B are subsets of U.

Lemma 2.4. *For any densification D,*

$$\begin{aligned}&\Pr\big(D(A)_i = D(B)_i | i \in N_{A,B}\big)\\&= \Pr\big(OPH(A)_i = OPH(B)_i | i \in N_{A,B}\big)\\&= J(A, B).\end{aligned}$$

This is just Lemma 2 in [Li et al., 2012] paraphrased using our notation. For completeness, we give the proof in Appendix.

The next lemma presents a critical property of a consistent densification scheme. Specifically, for any given simultaneously empty bin i, the assignment of i should mirror the collision probability of a simultaneously non-empty bin j. When that happens, we say that i *replicates* j.

Lemma 2.5. *Let i be a bin in $E_{A,B}$. For any consistent densification scheme D, there exists $j \in N_{A,B}$ such that*

$$\mathbb{1}\big(D(A)_i = D(B)_i\big) = \mathbb{1}\big(OPH(A)_j = OPH(B)_j\big).$$

Proof. For $i \in E_{A,B}$, let j be a bin sampled from $N_{A,B} = N_{OPH(A)} \cup N_{OPH(B)}$ using the same consistent sampling process of D. We first show $D(A)_i = D(B)_i$ if and only if $OPH(A)_j = OPH(B)_j$ by considering two cases:

- If $j \in N_{OPH(A)} \cap N_{OPH(B)}$ then both $f_{OPH(A)}(i)$ and $f_{OPH(B)}(i)$ are equal to j by consistency. Hence, $D(A)_i = OPH(A)_j$ and $D(B)_j = OPH(B)_j$. Hence, $\mathbb{1}\big(D(A)_i = D(B)_i\big) = \mathbb{1}\big(OPH(A)_j = OPH(B)_j\big)$ in this case.
- If $j \notin N_{OPH(A)} \cap N_{OPH(B)}$, we may assume without loss of generality that $j \in N_{OPH(A)} \setminus N_{OPH(B)}$. Then $OPH(A)_j \neq e = OPH(B)_j$. Moreover, $f_{OPH(A)}(i) = j$ by consistency. Therefore, $D(A)_i = OPH(A)_j$. Since $j \notin N_{OPH(B)}$,

$OPH(B)_l \neq OPH(A)_j$ for all $l \in N_{OPH(B)}$. Therefore, $\mathbb{1}\big(D(A)_i = D(B)_i\big) = 0 = \mathbb{1}\big(OPH(A)_j = OPH(B)_j\big)$.

The lemma follows from these two cases. ■

From Lemma 2.4 and Lemma 2.5, we have

Lemma 2.6. *If D is a consistent densification, for all $1 \leq i \leq k$,* $\Pr\big(D(A)_i = D(B)_i\big) = J(A,B)$.

The next lemma says that consistency is a condition that every densification must satisfy to create sketches having LSH property.

Lemma 2.7. *If D is a not consistent densification, there exists $1 \leq i \leq k$ and $A, B \subseteq U$ such that* $\Pr\big(D(A)_i = D(B)_i\big) < J(A,B)$.

Proof. First we show that for any densification D, $\Pr\big(D(A)_i = D(B)_i\big) \leq J(A,B)$. We consider two cases:

- For $i \in N_{A,B}$, by Lemma 2.4,
$$\Pr\big(D(A)_i = D(B)_i \mid i \in N_{A,B}\big) = J(A,B).$$
- For $i \in E_{A,B}$,
$$\Pr\big(D(A)_i = D(B)_i \mid i \in E_{A,B}\big) = \begin{cases} J(A,B), & \text{if } f_{OPH(A)}(i) = f_{OPH(B)}(i) \\ 0, & \text{otherwise.} \end{cases}$$
Therefore, we get $\Pr\big(D(A)_i = D(B)_i\big) \leq J(A,B)$.

Since D is not consistent, there exist $S \subseteq T \subset [k]$, $i \in [k] \setminus T$ such that $j \in S$ is sampled from T but j is not sampled from S in the process of filling bin i. Clearly, we can construct sets A, B large enough such that $N_{OPH(A)} = S$ and $N_{OPH(B)} = T$ with positive probability. Since $f_{OPH(B)}(i) = j \neq f_{OPH(A)}(i)$,

$$\Pr\big(D(A)_i = D(B)_i | N_{OPH(A)} = S, N_{OPH(B)} = T\big)$$

is equal to 0.

If we modify D so that $f_{OPH(A)}(i) = j$ and keep all other functions fixed, the above probability will increase to $J(A,B)$. Since the event $N_{OPH(A)} = S, N_{OPH(B)} = T$ occurs with a non-zero probability, the overall probability will increase. Hence, $\Pr\big(D(A)_i = D(B)_i\big) < J(A,B)$ before modification. ■

Combining Lemma 2.6 and Lemma 2.7 immediately gives Theorem 2.3.

3 A FASTER DENSIFICATION SCHEME

In this section we present a new densification scheme and prove Theorem 1.1. In particular, we show that our densification scheme takes $O(k \log k)$ expected time (Lemma 3.1), produces sketches with LSH property (Lemma 3.2) and has smaller variance than the one proposed in [Shrivastava, 2017] (Lemma 3.4).

Let $h_0, h_1, \ldots : [k] \to (0, k]$ be random hash functions mapping every element $1 \leq i \leq n$ to an independent uniform random value in $(0, k]$. For each $i \in [k]$, let

$$S_i^\alpha = \{j \in [k] : \lceil h_\alpha(j) \rceil = i\}.$$

The sets S_i^α can be viewed as a partition of the range space of h_α into k equal bins. Then for each h_α, define $g_\alpha : [k] \to [k] \cup e$ as follows:

$$g_\alpha(j) = \begin{cases} i, & \text{if } j = \arg\min_{j \in S_i} h_\alpha(j) \\ e, & \text{otherwise.} \end{cases}$$

In words, $g_\alpha(j)$ takes value i if and only if $h_\alpha(j)$ attains the smallest value in bin i. Our densification process is given in Algorithm 1.

Algorithm 1 Faster Densification

```
input: $v, \{h_\alpha\}$
output: $D(v)$
 1: for $j \in N_v$ do
 2:     $D(v)_j \leftarrow v_j$
 3: $E \leftarrow E_v$
 4: $\alpha \leftarrow 0$
 5: while $E \neq \emptyset$ do
 6:     for $j \in N_v$ do
 7:         if $g_\alpha(j) = i \in E$ then
 8:             $D(v)_i \leftarrow v_j$
 9:             $E \leftarrow E \setminus \{i\}$
10:             if $E = \emptyset$ then break
11:     $\alpha \leftarrow \alpha + 1$
12: return $D(v)$
```

At a high level, the algorithm fills empty bins in E_v with values from bins in N_v using a family of hash functions. In each round α, a hash function h_α (or equivalently the corresponding mapping function g_α) is utilized. In particular, the value of bin $j \in N_v$ is copied to bin $i \in E_v$ using $\{g_\alpha\}$ if $g_\alpha(j) = i$ and i is empty at the beginning of round α. A set E of the currently empty bins is maintained. The algorithm terminates after the round when there are no more empty bins, i.e., $E = \emptyset$.

Next we give the running time and correctness analysis of our densification.

3.1 RUNNING TIME

Lemma 3.1. *The proposed densification in Algorithm 1 takes $O(k \log k)$ time in expectation.*

Proof. Let x be the number of empty bins that are mapped to by some hash functions in $\{h_\alpha\}$ at any step of our algorithm. The probability that an empty bin is found (the probability that $h_\alpha(i)$ maps to an empty bin among k bins) is x/k. Therefore, the expected time that an empty bin is found, given x, is k/x. The total expected number of steps until all bins are mapped to by some hash function is:

$$\sum_{x=1}^{|E_v|} \frac{k}{x} = kH_{|E_v|},$$

where $H_{|E_v|}$ is the harmonic number of order $|E_v|$. After all bins are found, there are at most k more steps in the current round. Hence the total expected densification time is:

$$k + kH_{|E_v|}.$$

Since $H_{|E_v|} \leq H_k = O(\log k)$, the running time is $O(k \log k)$. ■

3.2 LSH PROPERTY

Lemma 3.2. *Let D be densification produced by Algorithm 1. Then*

$$\Pr\left(D(A)_i = D(B)_i\right) = J(A, B)$$

for any $A, B \subseteq U$.

Proof. By Theorem 2.3, it suffices to show that D is a consistent densification. Let v be a vector in U_e^k. For each $i \in E_v$, let $f_v(i) = j \in N_v$ be the bin in N_v from which i is assigned value according to D. Then $j = g_\alpha^{-1}(i)$ where α is the earliest round such that $i \in g_\alpha(N_v)$. Let u be another vector in U_e^k such that $N_u \subseteq N_v$ and assume that $j \in N_u$. Since $N_u \subseteq N_v$, the earliest round such that $i \in g_\alpha(N_v)$ must also be α. Therefore, $f_u(i) = j$ as desired. ■

3.3 VARIANCE

We give a variance analysis for our densification scheme. Most of the analysis is analogous for those presented in [Shrivastava and Li, 2014b] and [Shrivastava, 2017]. Recall that the estimator of $J(A, B)$ after densifying is given by

$$\tilde{J} = \frac{1}{k} \sum_{i \in [k]} \mathbb{1}\left(D(A)_i = D(B)_i\right).$$

Define the events M_i^N and M_i^E as:

$$M_i^N = \mathbb{1}\left(i \in N_{A,B} \text{ and } D(A)_i = D(B)_i\right),$$
$$M_i^E = \mathbb{1}\left(i \in E_{A,B} \text{ and } D(A)_i = D(B)_i\right).$$

In words, $M_i^N = 1$ if and only if i is simultaneously nonempty and matched after densification, and $M_i^E = 1$ if and only if i is simultaneously empty and matched after densification.

With respect to the above events, $\tilde{J}$ can be written as

$$\frac{1}{k} \sum_{i \in [k]} \left(M_i^N + M_i^E\right).$$

The variance is then

$$\mathrm{Var}\left(\tilde{J}\right) = \mathrm{E}\left[\left(\frac{1}{k} \sum_{i \in [k]} \left(M_i^N + M_i^E\right)\right)^2\right] - J^2.$$

We will use m to denote the event $|N_{A,B}| = m$ and let

$$F(m) = \mathrm{E}\left[\left(\frac{1}{k} \sum_{i \in [k]} \left(M_i^N + M_i^E\right)\right)^2 \middle| m\right].$$

By linearity of expectation,

$$k^2 F(m) = \mathrm{E}\left[\sum_{i \neq j} M_i^N M_j^N \middle| m\right] + \mathrm{E}\left[\sum_{i \neq j} M_i^N M_j^E \middle| m\right]$$
$$+ \mathrm{E}\left[\sum_{i \neq j} M_i^E M_j^E \middle| m\right] + \mathrm{E}\left[\sum_{i} \left[(M_i^N)^2 + (M_i^E)^2\right] \middle| m\right]$$

Since M_i^N and M_i^E are indicator variables,

$$\mathrm{E}\left[\sum_i \left[\left(M_i^N\right)^2 + \left(M_i^E\right)^2\right] \middle| m\right] = kJ.$$

For $i, j \in N_{A,B}$, $M_i^N M_j^N = 1$ with probability $J\hat{J}$, where

$$\hat{J} = \frac{|A \cap B| - 1}{|A \cup B| - 1}.$$

Therefore,

$$\mathrm{E}\left[\sum_{i \neq j} M_i^N M_j^N \middle| m\right] = m(m-1)J\hat{J}.$$

For $i \in N_{A,B}$ and $j \in E_{A,B}$, $M_i^N M_j^E = 1$ with probability $J\hat{J}$ if j replicates $i' \neq i$ and $M_i^N M_j^E = 1$ with probability J if j replicates i. Hence,

$$\mathrm{E}\left[\sum_{i\neq j} M_i^N M_j^E \middle| m\right] = 2m(k-m)\left(\frac{J}{m} + \frac{(m-1)J\hat{J}}{m}\right).$$

Similarly, for $i,j \in E_{A,B}$, $M_i^E M_j^E = 1$ with probability $J\hat{J}$ if i and j replicate different bins in $N_{A,B}$ and $M_i^N M_j^E = 1$ with probability J if they replicate the same bin. Therefore,

$$\mathrm{E}\left[\sum_{i\neq j} M_i^E M_j^E \middle| m\right] = (k-m)(k-m-1)\left(pJ + (1-p)J\hat{J}\right),$$

where p is the probability that two simultaneously empty bins replicate the same non-empty bin in the densification process. Since $pJ - pJ \cdot \hat{J} = pJ(1-\hat{J}) \geq 0$, $\mathrm{E}\left[\sum_{i\neq j} M_i^E M_j^E \middle| m\right]$ increases as p increases. Since the other terms are independent with p, $F(m)$ also increases as p increases. The next lemma gives an upper-bound on p.

Lemma 3.3. *In Algorithm 1, the probability that two simultaneously empty bins replicate the same simultaneously non-empty bin, given $|N_{A,B}| = m$, is*

$$\left(1 - \frac{1}{k}\right)^{m-1} \frac{1}{m} < \frac{1}{m}.$$

Proof. Notice that two simultaenously empty bins $i, j \in E_{A,B}$ replicate different bins if they are filled in the same round. The reason is $g_\alpha(i) = g_\alpha(j) = l$ implies $\lceil h_\alpha(l) \rceil$ is equal to both i and j, which is a contradiction.

Assume i is filled in round α and j is not filled before the α. The probaility that j is not filled in the same round is the probability that $h_\alpha(l)$ does not map to bin j for all $l \in N_{A,B} \setminus \{\lceil h_\alpha(i) \rceil\}$. This probability equals $\left(1 - \frac{1}{k}\right)^{m-1}$.

Conditioning on the event that i and j are filled in different rounds, we may assume that i is filled before j from bin l. The probability that j is filled from l is exactly $\frac{1}{m}$. To see this, observe that every bin in $N_{A,B}$ has an equal chance of being copied to j. ■

The values of p for densification shemes in [Shrivastava and Li, 2014a, Shrivastava and Li, 2014b, Shrivastava, 2017] are $\frac{2}{m+1}, \frac{1.5}{m+1}, \frac{1}{m}$ respectively. By Lemma 3.3, we have:

Lemma 3.4. *Our densification routine achieves smaller variance than the one proposed in [Shrivastava, 2017].*

4 EXPERIMENTS

In this section we describe experiments and present results that highlight the advantages of the densification routine proposed in the paper. These experiments closely resemble those in [Shrivastava, 2017]. We implemented three densification routines h^+ [Shrivastava and Li, 2014b], h^* [Shrivastava, 2017] and ours, denoted here by h^o. We could not find any code for the previous methods (the link was broken), but we implemented these methods, faithfully following the pseudo-code provided in these papers. The code was implemented in C++ and we use the popular MurmurHash[1] for all the methods. All the experiments were performed on a laptop (MacBook Pro 2015, i7 4770HQ 2.2 GHz, 16 GB RAM).

4.1 VARIANCE AND TIME FOR PAIRS

Table 1: Pair statistics including similarity and sparsity

	$\|S_1\|$	$\|S_2\|$	J
Pair 1	195	106	0.535714
Pair 2	649	452	0.045584
Pair 3	307	349	0.863636
Pair 4	303	156	0.207895

In this experiment, we chose four pairs of words from News20, represented as vectors using the term-document representation. These pairs were of varying Jaccard similarity, we ran each experiment 2000 times. The sizes of these word pairs and their Jaccard similarities are given in Table 1. We plot the mean squared error of h^+, h^* and h^o for several values of k. As a baseline, we estimate Jaccard similarity straight from one permutation hashing without any densification routine as in [Li et al., 2012]. This is a valid baseline because all densification routines copy values from non empty bins to empty bins. Hence, without using any further information, the variance of the estimate of the Jaccard similarity can only increase after densification. We note that the baseline used in [Shrivastava, 2017] is the theoretical variance of vanilla minwise hashing algorithm and is different from ours. Since all the routines are unbiased, the mean squared error corresponds to the variance of the estimators. The results of these experiments are shown in Figure 1.

We also noticed the effect of k on running time while doing these experiments. We give one such plot in Figure 2 for one of the word pairs. All plots we generated show a similar behavior. This plot shows the quadratic scaling

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

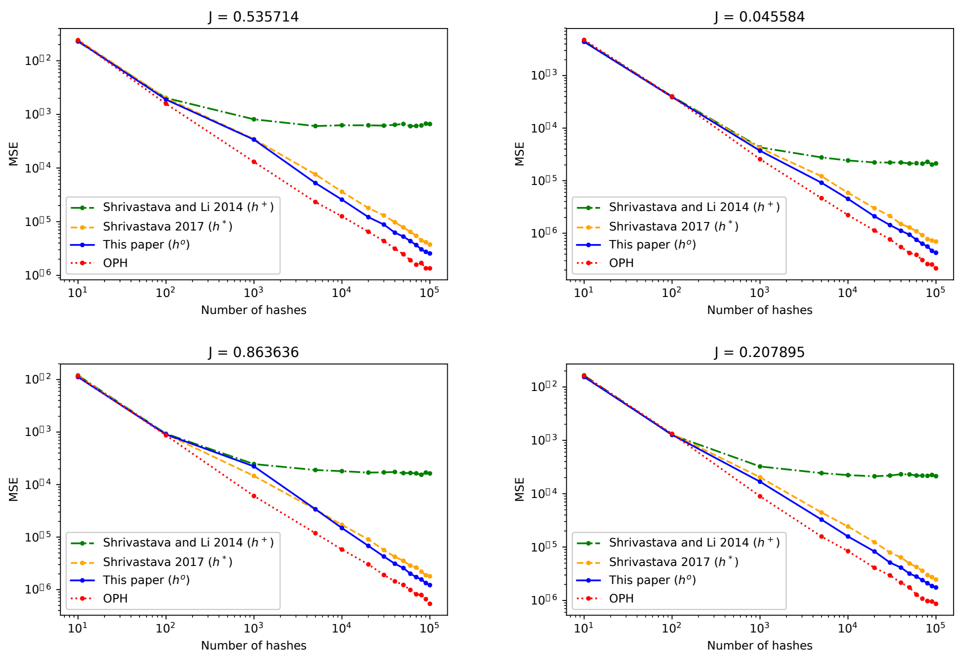

Figure 1: **Average MSE** in Jaccard Similarity Estimation as a function of (k). Estimates are averaged over 2000 repetitions. The accuracy of the proposed method is better than previous densification methods, as shown in theory. OPH is the error without any densification, but the sketch so produced does not have LSH property.

with k of h^* and nearly linear scaling of the proposed method.

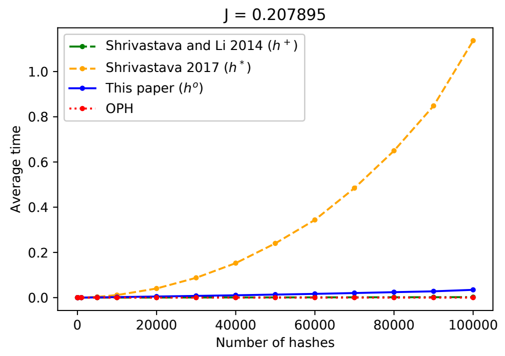

Figure 2: Time (in seconds) taken by various sketching routines for a word pair with similarity 0.207895, averaged over 2000 repetitions. Similar plots generated for other word pairs show similar behavior.

Conclusion 1 The variance h^o is at least as good as previous densification routines. The variance of h^*, h^o is much better than h^+; and h^o is better than h^* as k increases.

Conclusion 2 The running time of h^o scales nearly linearly with k.

4.2 TIME FOR SKETCHING FULL DATASETS

In this experiment, we run all the densification routines on three publicly available text datasets: News20, URL and RCV1. For the URL dataset, we only take the first 100000 samples. The details of the data sets are given in Table 4. We note the running time from start to finish (except the data loading time) for various values of k in Table 2. We clearly see the faster running times for our densification routine for large k. Since the running time depends on machines and code optimizations, we also give number of hash evaluations by the routines h^* and h^o incurred in the densification step. This is independent of the particular implementation. These numbers are normalized for each dataset and value of k such that the entry corresponding to h^o is always 1. We note that h^+ doesn't need any hash evaluations and hence, we do not include it in this table.

Conclusion 1 The time required by h^o scales nearly linearly with k whereas h^* performs increasingly worse as k increases. Since most datasets found in practice often contain many sparse sets, the performance of h^* is worse than h^o when the value of k is in the range (several thousands) that is often used in practice.

Conclusion 2 The performance gains achieved by h^o compared to h^* are larger for sparser data.

Conclusion 3 Compared to h^* that achieves state-of-

Table 2: Time (in seconds) required to compute $k = \{100, 1000, 10000\}$ hashes using the two existing densification routines and the proposed one, h^* can be much slower than h^o. While h^+ is very fast, it has a huge variance as discussed in [Shrivastava, 2017] and seen from the variance experiment in this paper.

	RCV1			URL			News20		
k	h^+	h^*	h^o	h^+	h^*	h^o	h^+	h^*	h^o
10^2	0.04	0.29	0.32	0.20	0.96	1.96	0.10	0.17	0.4
10^3	0.19	7.05	2.21	0.82	23.08	11.93	0.32	3.53	2.94
10^4	2.20	458.91	24.23	11.11	971.84	124.97	2.22	141.39	24.7

Table 3: Number of hash evaluations for computing $k = \{100, 1000, 10000\}$ hashes for h^* and h^o. We have normalized each combination of dataset and k so that the number of evaluations for h^o is one.

	RCV1		URL		News20	
k	h^*	h^o	h^*	h^o	h^*	h^o
10^2	0.41	1	0 .09	1	0.19	1
10^3	2.92	1	1.14	1	0.78	1
10^4	22.72	1	8.95	1	6.15	1

Table 4: Data statistics

Data	**Avg. non-zeros**	**Dim.**	**Samples**
RCV1	73	47,236	20,242
URL	115	3,231,961	100,000
News20	402	1,355,191	19,996

the-art variance, h^o is $6\times - 19\times$ faster for $k = 10^4$ (while always achieving a lower variance according to experiments carried on Section 4.1).

5 DISCUSSION

An undesirable property of our densification and the one given in [Shrivastava, 2017] is that the running time can be unbounded. This happens when the hash functions from the non-empty bins do not fully map into empty bins in our case (or the opposite direction in the case of [Shrivastava, 2017]). The issue, however, can be circumvented with a small cost in the variance guarantee. Specifically, we can restrict the number of hash functions described at the beginning of this section to k and add an extra k hash functions at the end to ansure that all the bins are completely filled. In partiticular, for $1 \le \alpha \le k$ the hash funcitons remain unchanged, and for $k+1 \le \alpha \le 2k$, $h_\alpha(i)$ maps a bin i in $[k]$ to the range $(\alpha - k, \alpha - k + 1]$. Hence, all hash functions are mapped to the same bin in the last k rounds. The non-empty bin whose hash attains the minimum value is chosen for reasignment. Since each bin i is guaranteed to be filled in round $k + i$, all bins are filled at the end.

With a more involved analysis, the running time can be proven to be $O(k \log k)$. It can also be seen that this modified densification is consistent. The variance, however, increases as it still remains under $\frac{1}{m}$. To see this, notice that if two bins are filled in the last k rounds, their reasigments are completely independent. Hence, we do not have the guarantee that two bins filled in the same round replicate different non-empty bins.

Another drawback of our densification is the use of multiple hash functions. It is not hard to see that, given any set A, with probability at least $1 - \frac{1}{k}$ we only need $\frac{k \log k}{|A|}$ hash functions to fill A. If we apply the above modification, the number of hash functions needed is $2k$. This can be avoided by utilizing a mixed tabulation in the manner as proposed in [Dahlgaard et al., 2017]. In particular, the mixed tabulation h has size $2k \log k$, and takes on input a pair (α, i). In other words, our algorithm remains unchanged after replacing $h_\alpha(i)$ with $h(\alpha, i)$. By [Dahlgaard et al., 2015], the keys in $\{0, \ldots, \lfloor \frac{k \log k}{|A|} \rfloor\} \times A$ are mapped independently with high probability, preserving our results.

References

[Bayardo et al., 2007] Bayardo, R. J., Ma, Y., and Srikant, R. (2007). Scaling up all pairs similarity search. In *Proceedings of the 16th international conference on World Wide Web*, pages 131–140. ACM.

[Broder, 1997] Broder, A. Z. (1997). On the resemblance and containment of documents. In *Compression and complexity of sequences 1997. proceedings*, pages 21–29. IEEE.

[Broder et al., 1997] Broder, A. Z., Glassman, S. C., Manasse, M. S., and Zweig, G. (1997). Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29(8):1157–1166.

[Chen and Shrivastava, 2016] Chen, B. and Shrivastava, A. (2016). Revisiting winner take all (WTA) hashing for sparse datasets. *CoRR*, abs/1612.01834.

[Chum et al., 2007] Chum, O., Philbin, J., Isard, M., and Zisserman, A. (2007). Scalable near identical image and shot detection. pages 549–556.

[Chum et al., 2008] Chum, O., Philbin, J., and Zisserman, A. (2008). Near duplicate image detection: min-hash and tf-idf weighting.

[D. Ondov et al., 2016] D. Ondov, B., J. Treangen, T., Melsted, P., B. Mallonee, A., Bergman, N., Koren, S., and M. Phillippy, A. (2016). Mash: Fast genome and metagenome distance estimation using minhash. *Genome Biology*, 17.

[Dahlgaard et al., 2015] Dahlgaard, S., Knudsen, M. B. T., Rotenberg, E., and Thorup, M. (2015). Hashing for statistics over k-partitions. In *Proc. 56th IEEE Symposium on Foundations of Computer Science*, pages 1292–1310. ACM.

[Dahlgaard et al., 2017] Dahlgaard, S., Knudsen, M. B. T., and Thorup, M. (2017). Fast similarity sketching. In *Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on*, pages 663–671. IEEE.

[Henzinger, 2006] Henzinger, M. (2006). Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 284–291. ACM.

[Indyk and Motwani, 1998] Indyk, P. and Motwani, R. (1998). Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM.

[Li et al., 2012] Li, P., Owen, A., and Zhang, C.-H. (2012). One permutation hashing. In *Advances in Neural Information Processing Systems*, pages 3113–3121.

[Li et al., 2011] Li, P., Shrivastava, A., Moore, J. L., and König, A. C. (2011). Hashing algorithms for large-scale learning. In *Advances in neural information processing systems*, pages 2672–2680.

[Manasse et al., 2010] Manasse, M., McSherry, F., and Talwar, K. (2010). Consistent weighted sampling.

[Shi et al., 2018] Shi, Y., Cost, R., Perlich, C., Hook, R., Martin, W., Han Williams, M., Moynihan, J., McCarthy, P., Lenz, P., and Daniel-Weiner, R. (2018). Audience size forecasting: Fast and smart budget planning for media buyers. pages 744–753.

[Shrivastava, 2017] Shrivastava, A. (2017). Optimal densification for fast and accurate minwise hashing. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3154–3163, International Convention Centre, Sydney, Australia. PMLR.

[Shrivastava and Li, 2014a] Shrivastava, A. and Li, P. (2014a). Densifying one permutation hashing via rotation for fast near neighbor search. In *International Conference on Machine Learning*, pages 557–565.

[Shrivastava and Li, 2014b] Shrivastava, A. and Li, P. (2014b). Improved densification of one permutation hashing. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, UAI'14, pages 732–741, Arlington, Virginia, United States. AUAI Press.