Optimal Densification for Fast and Accurate Minwise Hashing

Anshumali Shrivastava [1]

Abstract

Minwise hashing is a fundamental and one of the most successful hashing algorithm in the literature. Recent advances based on the idea of densification (Shrivastava & Li, 2014a;c) have shown that it is possible to compute k minwise hashes, of a vector with d nonzeros, in mere $(d + k)$ computations, a significant improvement over the classical $O(dk)$. These advances have led to an algorithmic improvement in the query complexity of traditional indexing algorithms based on minwise hashing. Unfortunately, the variance of the current densification techniques is unnecessarily high, which leads to significantly poor accuracy compared to vanilla minwise hashing, especially when the data is sparse. In this paper, we provide a novel densification scheme which relies on carefully tailored 2-universal hashes. We show that the proposed scheme is variance-optimal, and without losing the runtime efficiency, it is significantly more accurate than existing densification techniques. As a result, we obtain a significantly efficient hashing scheme which has the same variance and collision probability as minwise hashing. Experimental evaluations on real sparse and high-dimensional datasets validate our claims. We believe that given the significant advantages, our method will replace minwise hashing implementations in practice.

1. Introduction and Motivation

Recent years have witnessed a dramatic increase in the dimensionality of modern datasets. (Weinberger et al., 2009) show dataset with 16 trillion (10^{13}) unique features. Many studies have shown that the accuracy of models keeps climbing slowly with exponential increase in dimensionality. Large dictionary based representation for images, speech, and text are quite popular (Broder, 1997; Fetterly et al., 2003). Enriching features with co-occurrence information leads to blow up in the dimensionality. 5-grams are common for text representations. With vocabulary size of 10^6, 5-grams representation requires dimensionality of 10^{30}. Representing genome sequences with features consisting of 32-contiguous characters (or higher) (Ondov et al., 2016) leads to around $4^{32} = 2^{64}$ dimensions.

To deal with the overwhelming dimensionality, there is an increased emphasis on the use of hashing algorithms, such as minwise hashing. Minwise hashing provides a convenient way to obtain a compact representation of the data, without worrying about the actual dimensionality. These compact representations are directly used in large scale data processing systems for a variety of tasks.

Minwise hashing is defined for binary vectors. Binary vectors can also be equivalently viewed as sets, over the universe of all the features, containing only attributes corresponding to the non-zero entries. Minwise hashing belongs to the *Locality Sensitive Hashing (LSH)* family (Broder et al., 1998; Charikar, 2002). The method applies a random permutation (or random hash function) $\pi : \Omega \to \Omega$, on the given set $S \subset \Omega$, and stores the minimum value after the permutation mapping. Formally,

$$h_\pi(S) = \min(\pi(S)). \tag{1}$$

Given sets S_1 and S_2, it can be shown by elementary probability arguments that

$$Pr(h_\pi(S_1) = h_\pi(S_2)) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = R. \tag{2}$$

The quantity

$$R = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = \frac{a}{f_1 + f_2 - a}, \tag{3}$$

is the well known *Jaccard Similarity* (or resemblance) R which is the most popular similarity measure in information retrieval applications (Broder, 1997).

The probability of collision (equality of hash values), under minwise hashing, is equal to the similarity of interest R. This particular property, also known as the *LSH property* (Indyk & Motwani, 1998; Charikar, 2002), makes minwise hash functions h_π suitable for creating hash buckets,

[1]Rice University, Houston, TX, USA. Correspondence to: Anshumali Shrivastava <anshumali@rice.edu>.

Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

which leads to sublinear algorithms for similarity search. Because of this same LSH property, minwise hashing is a popular indexing technique for a variety of large-scale data processing applications, which include duplicate detection (Broder, 1997; Henzinger, 2006), all-pair similarity (Bayardo et al., 2007), temporal correlation (Chien & Immorlica, 2005), graph algorithms (Buehrer & Chellapilla, 2008; Chierichetti et al., 2009; Najork et al., 2009), hashing time series (Luo & Shrivastava, 2017), and more. It was recently shown that the *LSH property* of minwise hashes can be used to generate kernel features for large-scale learning (Li et al., 2011).

Minwise hashing is known to be theoretical optimal in many scenarios (Bavarian et al., 2016). Furthermore, it was recently shown to be provably superior LSH for angular similarity (or cosine similarity) compared to widely popular Signed Random Projections (Shrivastava & Li, 2014b). These unique advantages make minwise hashing arguably the strongest hashing algorithm both in theory and practice.

Hashing Cost is Bottleneck: The first step of algorithms relying on minwise hashing is to generate, some large enough, k minwise hashes (or fingerprints) of the data vectors. In particular, for every data vector x, $h_i(x)$ $\forall i \in \{1, 2, ..., k\}$ is repeatedly computed with independent permutations (or hash functions). These k hashes are used for a variety of data mining tasks such as cheap similarity estimation, indexing for sub-linear search, kernel features for large scale learning, etc. Computing k hashes of a vector x with traditional minwise hashing requires $O(dk)$ computation, where d is the number of non-zeros in vector x. This computation of the multiple hashes requires multiple passes over the data. The number of hashes required by the famous LSH algorithm is $O(n^\rho)$ which grows with the size of the data. (Li, 2015) showed the necessity of around 4000 hashes per data vector in large-scale learning. Note, advances in efficient weighted minwise hashing by (Shrivastava, 2016) is not applicable for very sparse unweighed minwise hashing.

Other Related Fast Sketches are not LSH: Two notable techniques for estimating Jaccard Similarity are: 1) bottom-k sketches and 2) one permutation hashing (Li et al., 2012). Although these two sketches are cheap to compute, they do not satisfy the key *LSH property* and therefore are unsuitable for replacing minwise hashing (Shrivastava & Li, 2014a;c). There are also substantial empirical evidence that using these (non-LSH) sketches for indexing leads to a drastic bias in the expected behavior, leading to poor accuracy.

The Idea of "Densified" One Permutation Hashing: Recently, (Shrivastava & Li, 2014a) showed a technique for densifying sparse sketches from one permutation hashing which provably removes the bias associated with one permutation hashing. Please see Section 3.3 for details. The scheme is much cheaper than the later discovered alternatives which rely on continued sampling (or multiple hashes) until no empty bins are left (Haeupler et al., 2014; Dahlgaard et al., 2017). (Shrivastava & Li, 2014a) was the first success in creating efficient hashing scheme which satisfies the *LSH property* analogous to minwise hashing and at the same time the complete process only requires $O(d+k)$ computations instead of the traditional bottleneck of $O(dk)$, and only one hash function.

Current Densification is Inaccurate For Very Sparse Datasets: The densification process although efficient and unbiased was shown to have unnecessarily higher variance. It was shown in (Shrivastava & Li, 2014c) that the traditional "densification" lacks sufficient randomness. It was further revealed that densification could be provably improved by using k extra random bits. An improved variance was associated with a significant performance gain in the task of near-neighbor search. In this work, we show that even the improved densification scheme is far from optimal. The findings of (Shrivastava & Li, 2014c) leaves an open curiosity: What is the best variance that can be achieved with "densification" without sacrificing the running time? We close this by providing a variance-optimal scheme.

Our Contributions: We show that the existing densification schemes, for fast minwise hashing, are not only sub-optimal but, worse, their variances do not go to zero with increasing number of hashes. The variance with an increase in the number of hashes converges to a positive constant. This behavior implies that increasing the number of hashes after a point will lead to no improvement, which is against the popular belief that accuracy of randomized algorithms keeps improving with an increase in the number of hashes.

To circumvent these issues we present a novel densification scheme which has provably superior variance compared to existing schemes. We show that our proposal has the optimal variance that can be achieved by densification. Furthermore, the variance of new methodology converges to zero with an increase in the number of hashes, a desirable behavior absent in prior works. Our proposal makes novel use of 2-universal hashing which could be of independent interest in itself. The benefits of improved accuracy come with no loss in computational requirements, and our scheme retains the running time efficiency of the densification.

We provide rigorous experimental evaluations of existing solutions concerning both accuracy and running time efficiency, on real high-dimensional datasets. Our experiments validate all our theoretical claims and show significant improvement in accuracy, comparable to minwise hashing, with a significant gain in computational efficiency.

2. Important Notations and Concepts

Equation 2 (i.e. the *LSH Property*) leads to an estimator of Jacard Similarity R, using k hashes, defined by:

$$\hat{R} = \frac{1}{k}\sum_{i=1}^{k} \mathbf{1}(h_i(S_1) = h_i(S_2)). \tag{4}$$

Here $\mathbf{1}$ is the indicator function. In the paper, by variance, we mean the variance of the above estimator. Notations like $Var(h^+)$, will mean the variance of the above estimator when the h^+ is used as the hash function.

$[k]$ will denote the set of integers $\{1, 2, ..., k\}$. n denotes the number of points (samples) in the dataset. D will be used for dimensionality. We will use $\min\{S\}$ to denote the minimum element of the set S. A permutation $\pi : \Omega \to \Omega$ applied to a set S is another set $\pi(S)$, where $x \in S$ if and only if $\pi(x) \in \pi(S)$. Our hashing will generate k hashes h_i $i \in \{1, 2, ..., k\}$, generally from different bins. Since they all have same distribution and properties we will drop subscripts. We will use h and h^+ to denote the hashing schemes of (Shrivastava & Li, 2014a) and (Shrivastava & Li, 2014c) respectively.

3. Background: Fast Minwise Hashing via Densification

3.1. 2-Universal Hashing

Definitions: A randomized function $h_{univ} : [l] \to [k]$ is 2-universal if for all, $i, j \in [l]$ with $i \neq j$, we have the following property for any $z_1, z_2 \in [k]$ $Pr(h_{univ}(i) = z_1$ and $h_{univ}(j) = z_2) = \frac{1}{k^2}$

(Carter & Wegman, 1977) showed that the simplest way to create a 2-universal hashing scheme is to pick a prime number $p \geq k$, sample two random numbers a, b and compute $h_{univ}(x) = ((ax + b) \bmod p) \bmod k$

3.2. One Permutation Hashing and Empty Bins

It was shown in (Li et al., 2012; Dahlgaard et al., 2015) that instead of computing the global minimum in Equation 2, i.e., $h(S) = \min(\pi(S))$, an efficient way to generate k sketches, using one permutation, is to first bin the range space of π, i.e. Ω, into k disjoint and equal partitions followed by computing minimum in each bin (or partition).

Let Ω_i denote the i^{th} partition of the range space of π, i.e. Ω. Formally, the i^{th} one permutation hashes (OPH) of a set S is defined as

$$h_i^{OPH}(S) = \begin{cases} \min\{\pi(S) \cap \Omega_i\}, & \text{if } \{\pi(S) \cap \Omega_i\} \neq \phi \\ E, & \text{otherwise.} \end{cases} \tag{5}$$

An obvious computational advantage of this scheme is that it is likely to generate many hash values, at most k, and only requires one permutation π and only pass over the sets (or binary vectors) S. It was shown that for any two sets S_1 and S_2 we have a conditional collision probability similar to minwise hashing.

$$\text{Let} \quad E_i = \mathbf{1}\{h_i^{OPH}(S_2) = h_i^{OPH}(S_2) = E\} \tag{6}$$

$$Pr\big(h_i^{OPH}(S_1) = h_i^{OPH}(S_2) \mid E_i = 0\big) = R \tag{7}$$

$$\text{However, } Pr\big(h_i^{OPH}(S_1) = h_i^{OPH}(S_2) \mid E_i = 1\big) \neq R \tag{8}$$

Here E_i is an indicator random variable of the event that the i^{th} partition corresponding to both S_1 and S_2 are empty. See Figure 1

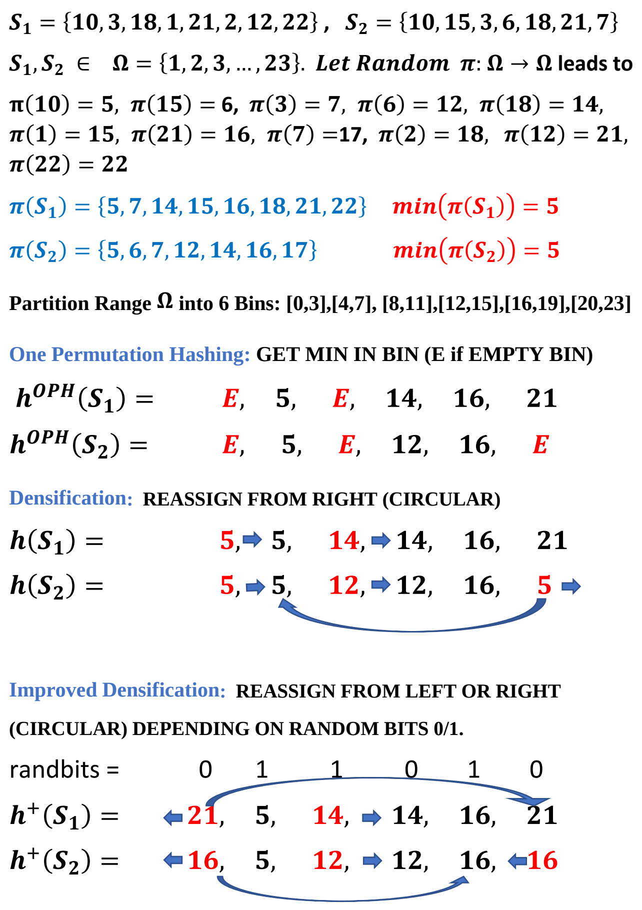
Any bin has a constant chance of being empty. Thus, there is a positive probability of the event $\{E_i = 1\}$, for any given pair S_1 and S_2 and hence for large datasets (big n) a constant fraction of data will consist of simultaneously empty bins (there are $n^2 \times k$ trials for the bad event $\{E_i = 1\}$ to happen). This fraction further increases significantly with the sparsity of the data and k, as both sparsity and k increases the probability of the bad event $\{E_i = 1\}$. See Table 4 for statistics of empty bins on real scenarios.

Unfortunately, whenever the outcome of the random permutation leads to simultaneous empty bins, i.e. event $E_i = 1$, the *LSH Property* is not valid. In fact, there is not sufficient information present in the simultaneous empty partitions for any meaningful statistics. Hence, one permutation hashing cannot be used as an LSH. Simple heuristics of handling empty bins as suggested in (Shrivastava & Li, 2014a) leads to a significant bias and it was shown both theoretically and empirically that this bias leads to significant deviation from the expected behavior of one permutation hashing when compared with minwise hashing. Thus, one permutation hashing although computationally lucrative is not a suitable replacement for minwise hashing.

3.3. The Idea of Densification

In (Shrivastava & Li, 2014a), the authors proposed “densification” or reassignment of values to empty bins by reusing the information in the non-empty bins to fix the bias of one permutation hashing. The overall procedure is quite simple. Any empty bin borrows the values of the closest non-empty bins towards the circular right (or left)[1]. See Figure 1 for an illustration. Since the positions of empty

[1]In (Shrivastava & Li, 2014a) they also needed an offset because the value of a hash in any bin was always reset between [0,k]. We do not need the offset if we use the actual values of $\pi(S)$.



$S_1 = \{10,3,18,1,21,2,12,22\}$, $S_2 = \{10,15,3,6,18,21,7\}$
$S_1, S_2 \in \Omega = \{1,2,3,\dots,23\}$. Let Random $\pi: \Omega \to \Omega$ leads to
$\pi(10)=5$, $\pi(15)=6$, $\pi(3)=7$, $\pi(6)=12$, $\pi(18)=14$, $\pi(1)=15$, $\pi(21)=16$, $\pi(7)=17$, $\pi(2)=18$, $\pi(12)=21$, $\pi(22)=22$
$\pi(S_1)=\{5,7,14,15,16,18,21,22\}$ $min(\pi(S_1))=5$
$\pi(S_2)=\{5,6,7,12,14,16,17\}$ $min(\pi(S_2))=5$
Partition Range Ω into 6 Bins: [0,3],[4,7], [8,11],[12,15],[16,19],[20,23]
One Permutation Hashing: GET MIN IN BIN (E if EMPTY BIN)
$h^{OPH}(S_1) =$ E, 5, E, 14, 16, 21
$h^{OPH}(S_2) =$ E, 5, E, 12, 16, E
Densification: REASSIGN FROM RIGHT (CIRCULAR)
$h(S_1) =$ 5, 5, 14, 14, 16, 21
$h(S_2) =$ 5, 5, 12, 12, 16, 5
Improved Densification: REASSIGN FROM LEFT OR RIGHT (CIRCULAR) DEPENDING ON RANDOM BITS 0/1.
randbits = 0 1 1 0 1 0
$h^+(S_1) =$ 21, 5, 14, 14, 16, 21
$h^+(S_2) =$ 16, 5, 12, 12, 16, 16

Figure 1. Illustration of One Permutaion Hashing (OPH) and the two existing Densification Schemes of (Shrivastava & Li, 2014a;c). Densification simply borrows the value from nearby empty bins. Different sets S_i with have different pattern of empty/nonempty bins. Since the pattern is random, the process is similar to random re-use of unbiased values, which satisfies LSH property in each bin (Shrivastava & Li, 2014c).

and non-empty bins were random, it was shown that densification (or reassignment) was equivalent to a stochastic reselection of one hash from a set of existing informative (coming from non-empty bins) hashes which have the LSH property. This kind of reassignment restores the *LSH property* and collision probability for any two hashes, after reassignment, is exactly same as that of minwise hashing.

The densification generates k hashes with the required *LSH Property* and only requires two passes over the one permutation sketches making the total cost of one permutation hashing plus densification $O(d+k)$. This was a significant improvement over $O(dk)$ with classical minwise hashing. $O(d+k)$ led to an algorithmic improvement over randomized algorithms relying on minwise hashing, as hash computation cost is bottleneck step in all of them.

3.4. Lack of Randomness in Densification

It was pointed out in (Shrivastava & Li, 2014c) that the densification scheme of (Shrivastava & Li, 2014a) has unnecessarily high variance. In particular, the probability of two empty bins borrowing the information of the same non-empty bin was significantly higher. This probability was due to poor randomization (load balancing) which hurts the variance. (Shrivastava & Li, 2014c) showed that infusing more randomness in the reassignment process by utilizing k extra random bits provably improves the variance. See Figure 1 for an example illustration of the method. The running time of the improved scheme was again $O(d+k)$ for computing k hashes. This improvement retains the required *LSH property*, however this time with improved variance. An improved variance led to significant savings in the task of near-neighbor search on real sparse datasets.

4. Issues with Current Densification

Our careful analysis reveals that the variance, even with the improved scheme, is still significantly higher. Worse, even in the extreme case when we take $k \to \infty$ the variance converges to a positive constant rather than zero, which implies that even with infinite samples, the variance will not be zero. This positive limit further increases with the sparsity of the dataset. In particular, we have the following theorem about the limiting variances of existing techniques:

Theorem 1 *Give any two finite sets $S_1, S_2 \in \Omega$, with $A = |S_1 \cup S_2| > a = |S_1 \cap S_2| > 0$ and $|\Omega| = D \to \infty$. The limiting variance of the estimators from densification and improved densification when $k = D \to \infty$ is given by:*

$$\lim_{k\to\infty} Var(h) = \frac{a}{A}\left[\frac{A-a}{A(A+1)}\right] > 0 \quad (9)$$

$$\lim_{k\to\infty} Var(h^+) = \frac{a}{A}\left[\frac{3(A-1)+(2A-1)(a-1)}{2(A+1)(A-1)} - \frac{a}{A}\right] > 0 \quad (10)$$

This convergence of variance to a constant value, despite infinite samples, of the existing densification is also evident in our experimental findings (see Figure 3) where we observe that the MSE (Mean Square Error) curves go flat with increasing k. Similar phenomenon was also reported in (Shrivastava & Li, 2014c). It should be noted that for classical minwise hashing the variance is $\frac{R(1-R)}{k} \to 0$ for any pair S_1 and S_2. Thus, current densification, although fast, loses significantly in terms of accuracy. We remove this issue with densification. In particular, we show in Theorem 2 that the limiting variance of the proposed optimal densification goes to 0. Our experimental findings suggests that the new variance is very close to the classical minwise hashing. In addition, the new densification retains the speed of existing densified hashing thereby achieving the best of the both worlds.

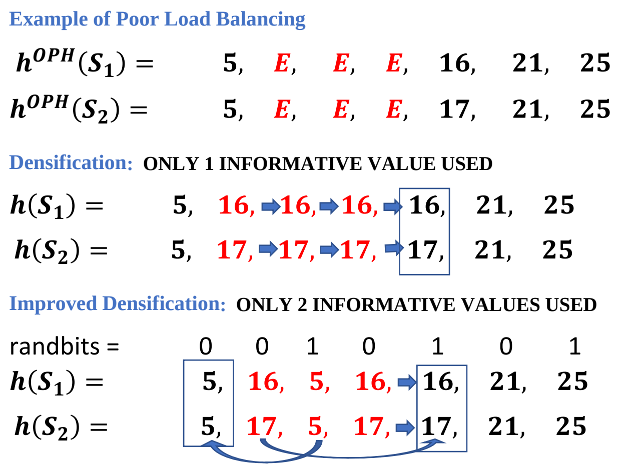

Example of Poor Load Balancing
h^OPH(S1) = 5, E, E, E, 16, 21, 25
h^OPH(S2) = 5, E, E, E, 17, 21, 25
Densification: ONLY 1 INFORMATIVE VALUE USED
h(S1) = 5, 16, 16, 16, 16, 21, 25
h(S2) = 5, 17, 17, 17, 17, 21, 25
Improved Densification: ONLY 2 INFORMATIVE VALUES USED
randbits = 0 0 1 0 1 0 1
h(S1) = 5, 16, 5, 16, 16, 21, 25
h(S2) = 5, 17, 5, 17, 17, 21, 25

Figure 2. Illustration of Poor Load Balancing in the existing Densification strategies. The re-assignment of bins is very local and its not uniformly distributed. Thus, we do not use the information in far off non-empty bins while densification.

5. Optimal Densification

We argue that even with the improved densification there is not enough randomness (or load balancing) in the re-assignment process which leads to reduced variance.

For given set S, the densification process reassigns every empty bin with a value from one of the existing non-empty bins. Note, the identity of empty and non-empty bins are different for different sets. To ensure the LSH property, the re-assignment should be consistent for any given pair of sets S_1 and S_2. In particular, as noted in (Shrivastava & Li, 2014a), given any arbitrary pair S_1 and S_2, whenever any given bin i is simultaneously empty, i.e. $E_i = 1$, the reassignment of this bin i should mimic the collision probability of one of the simultaneously non-empty bin j with $E_j = 0$. An arbitrary reassignment (or borrow) of values will not ensure this consistency across all pairs. We would like to point out that the reassignment of S_1 has no idea about S_2 or any other object in the dataset. Thus, ensuring the consistency is non-trivial. Although the current densification schemes achieve this consistency by selecting the nearest non-empty bin (as shown in (Shrivastava & Li, 2014c)), they lack sufficient randomness.

5.1. Intuition: Load Balancing

In Figure 2, observe that if there are many contiguous non-empty bins (Bins 2, 3 and 4), then with densification schemes h, all of them are forced to borrow values from the same non-empty bin (Bin 5 in the example). Even though there are other informative bins (Bins 1, 6 and 7), their information is never used. This local bias increases the probability (p) that two empty bins get tied to the same information, even if there are many other informative non-empty bins. Adding k random bits improves this to some extent by allowing load sharing between the two ends instead of one (Bins 1 and 5 instead of just 5). However, the load balancing is far from optimal. The locality of information sharing is the main culprit with current densification schemes. Note, the poor load balancing does not change the expectation but affects the variance significantly.

For any given pairs of vectors S_1 and S_2, let m be the number of simultaneous non-empty bins (out of k), i.e. $\sum_{i=1}^{k} E_i = k - m$. Note, m is a random variable whose value is different for every pair and depends on the outcome of random π. Formally, the variance analysis of (Shrivastava & Li, 2014c) reveals that the probability that any two simultaneous empty bin p and q ($E_p = E_q = 1$) reuses the same information is $\frac{2}{m+1}$ with the densification scheme h. This probability was reduced down to $\frac{1.5}{m+1}$ with h^+ by utilizing k extra random bits to promote load balancing.

$p = \frac{1.5}{m+1}$ is not quite perfect load balancing. In a perfect load balancing with m simultaneous non-empty bins, the probability of two empty bins hitting the same non-empty bins is at best $p = \frac{1}{m}$. Can we design a densification scheme which achieves this p while maintaining the consistency of densification and at the same time does not hurt the running time? It is not clear if such a scheme even exists. We answer this question positively by constructing a densification method with precisely all the above requirements. Furthermore we show that achieving $p = \frac{1}{m}$ is sufficient for having the limiting variance of zero.

5.2. Simple 2-Universal Hashing Doesn't Help

To break the locality of the information reuse and allow a non-empty bin to borrow information from any other far off bin consistently, it seems natural to use universal hashing. The hope is to have a 2-universal hash function (Section 3.1) $h_{univ} : [k] \to [k]$. Whenever a bin i is empty, instead of borrowing information from neighbors, borrow information from bin $h_{univ}(i)$. The hash function allows consistency across any two S_1 and S_2 hence preserves *LSH property*. The value of $h_{univ}(i)$ is uniformly distributed, so any bin is equally likely. Thus, it seems to break the locality on the first thought. If $h_{univ}(i)$ is also empty then we continue using $h_{univ}(h_{univ}(i))$ until we reach a non-empty bins whose value we re-use.

One issue is that of cycles. If $i = h_{univ}(i)$ (which has $\frac{1}{k}$ chance), then this creates a cycle and the assignment will go into infinite loop. A cycle can even occur if $h_{univ}(h_{univ}(i)) = i$ with both i and $h_{univ}(i)$ being empty. Note, the process runs until it finds a non-empty bin. However, cycles are not just our concern. Even if we manage to get away with cycles, this scheme does not provide the required load balancing.

A careful inspection reveals that there is a very significant chance that both i and $h_{univ}(i)$ to be empty for any given set S. Observe that if i and $h_{univ}(i)$ are both empty, then we are bound to reuse the information of the same non-empty bin for both empty bins i and $h_{univ}(i)$. We should note that we have no control over the positions of empty and non-empty bins. In fact, if no cycles happen then it is not difficult to show that the simple assignment using universal hashing is equivalent to the original densification h with the order of bins reshuffled using $h_{univ}(.)$. It has worse variance than h^+.

5.3. The Fix: Carefully Tailored 2-Universal Hashing

Algorithm 1 Optimal Densification

```
input  $k$ One Permutation Hashes $h^{OPH}[\,]$ of $S$.
input  $h_{univ}(.,.)$

Initialize $h^*[\,] = 0$
for $i = 1$ to $k$ do
  if $OPH[i] \neq E$ then
    $h^*[i] = h^{OPH}[i]$
  else
    $attempt = 1$
    $next = h_{univ}(i, attempt)$
    while $h^{OPH}[next] \neq E$ do
      $attempt++$
      $next = h_{univ}(i, attempt)$
    end while
    $h^*[i] = h^{OPH}[next]$
  end if
end for
RETURN $h^*[\,]$
```

It turns out that there is a way to use universal hashing that ensures no cycles as well as optimal load balancing. We describe the complete process in Algorithm 1. The key is to use a 2-universal hashing $h_{univ} : [k] \times \mathbb{N} \to [k]$ which takes two arguments: 1) The current bin id that needs to be reassigned and 2) the number of failed attempt made so far to reach a non-empty bin. This second argument ensures no infinite loops as it changes with every attempt. So even if we reach the same non-empty bin back (cycle), the next time we will visit a new set of bins. Also, even if both i and $j = h_{univ}(i, attempt_i)$ are empty, i and j are not bound to end to the same non-empty bin. This is because in the next attempt we seek bin value $h_{univ}(i, attempt_i + 1)$ for i which is independent of the $h_{univ}(j, attempt_j)$ due to 2-universality of the hash function h_{univ}. Thus, the probability that any two empty bins reuse the information of the same non-empty bin is $\frac{1}{m}$

5.4. Analysis and Optimality

We denote the final k hashes generated by the proposed densification scheme of Algorithm 1 using h^* (* for optimality). Formally, with the optimal densification h^*, we have the following:

Theorem 2

$$Pr\big(h^*(S_1) = h^*(S_2)\big) = \frac{|S_1 \cap S_2|}{|S_1 \cap S_2|} = R \qquad (11)$$

$$Var(h^*) = \frac{R}{k} + A\frac{R}{k^2} + B\frac{R\bar{R}}{k^2} - R^2 \qquad (12)$$

$$\lim_{k\to\infty} Var(h^*) = 0 \qquad (13)$$

where N_{emp} is the number of simultaneous empty bins between S_1 and S_2 and the quantities A and B are given by

$$A = \mathbb{E}\left[2N_{emp} + \frac{N_{emp}(N_{emp}-1)}{k - N_{emp}}\right]$$

$$B = \mathbb{E}\Bigg[(k - N_{emp})(k - N_{emp} - 1) + 2N_{emp}(k - N_{emp} - 1) + \frac{N_{emp}(N_{emp}-1)(k - N_{emp} - 1)}{k - N_{emp}}\Bigg]$$

Using the formula for $Pr(N_{emp} = i)$ from (Li et al., 2012), we can precisely compute the theoretical variance. The interesting part is that we can formally show that the variance of the proposed scheme is strictly superior compared to the densification scheme with random bits improvements.

Theorem 3

$$Var(h^*) \leq Var(h^+) \leq Var(h) \qquad (14)$$

Theorem 4 *Among all densification schemes, where the reassignment process for bin i is independent of the reassignment process of any other bin j, Algorithm 1 achieves the best possible variance.*

Note: The variance can be reduced if we allow correlations in the assignment process, for example if we force bin i and bin j to not pick the same bin during reassignments, this will reduce p beyond the perfectly random load balancing value of $\frac{1}{m}$. However, such tied reassignment will require more memory and computations for generating structured hash functions. Also, we use only one hash function (or permutation). If we allow multiple independent hash function then with additional computational and memory cost the variance can be further reduced.

5.5. Running Time

We show that the expected running time of our proposal, including all constants, is very similar to the running time of the existing densification schemes.

Given set S with $|S| = d$, we are interested in computing k hash values. The first step involves computing k one permutation hashes (or sketches) which only requires a single

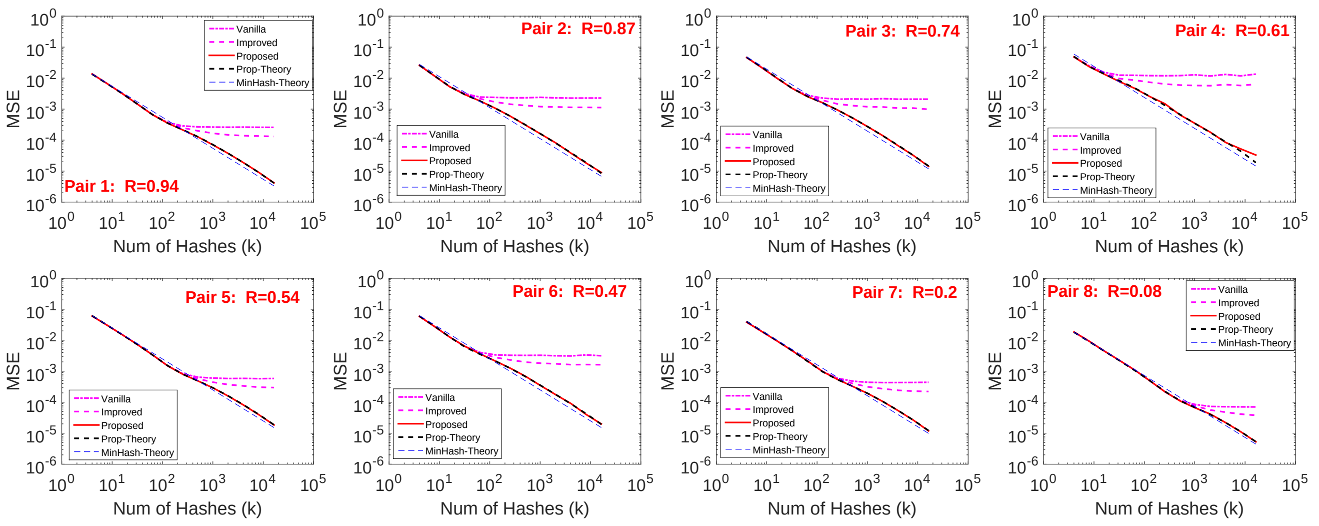


Figure 3. **Average MSE** in Jaccard Similarity Estimation with the Number of Hash Values (k). Estimates are averaged over 5000 repetitions. With Optimal densification the variance is very close to costly minwise hashing, which is significantly superior to existing densification schemes. Note the log scale of y-axis.

	Sim R	$\|S_1\|$	$\|S_2\|$
Pair 1	0.94	208	196
Pair 2	0.87	47	41
Pair 3	0.74	91	67
Pair 4	0.61	15	14
Pair 5	0.54	419	232
Pair 6	0.47	76	36
Pair 7	0.20	264	182
Pair 8	0.02	655	423

Table 1. Statistics of Pairs vectors (binary) with their Similarity and Sparsity (non-zeros).

pass over the elements of S. This takes $\max\{d,k\} \leq d+k$ time. Now the densification Algorithm 1 requires a for loop of size k and within each for loop, if the bin is empty, it requires an additional while loop. Let N_{emp} be the number of empty bins, and therefore $\frac{k-N_{emp}}{k}$ is the probability that the while loop will terminate in one iteration (next is not empty). Therefore, the expected number of iteration that each while loop will run is a binomial random variable with expectation $\frac{k}{k-N_{emp}}$. Thus, the expected running time of the algorithm is given by

$$E[\text{Running Time}] = d + 2k - N_{emp} + N_{emp}\frac{k}{k - N_{emp}}$$
$$\leq d + (r+2)k \text{ where } r = \frac{N_{emp}}{N_{not-emp}}$$

The quantity r, which is the ratio of number of empty bins to the number of non-empty bins, is generally very small. It is rarely more than 2 to 3 in practice. Observe that randomly throwing d items into k bins, the expected number of empty bins is $E[N_{emp}] \approx k(1-\frac{1}{k})^d \approx ke^{-\frac{d}{k}}$. Which makes $r \approx \frac{e^{-d/k}}{1-e^{-d/k}}$. The number of sketches is usually of the order of non-zeros. Even for very good concentration, the size of the sketches k is rarely much larger than the size of the set d. Even when k is 4 times d the value of r is approximately 3.5. Thus, the quantity r is negligible.

It should be further noted that the implementation cost of densification scheme h^+ is $d + 4k$ which in not very different from the cost of our proposal.

6. Evaluations

Our aim is to verify the theoretical claims of these papers empirically. We show that our proposal can replace minwise hashing for all practical purposes. To establish that, we focus on experiments with the following objectives:

1. Verify that our proposed scheme has a significantly better accuracy (variance) than the existing densification schemes. Validate our variance formulas.

2. Empirically quantify the impact of optimal variance in practice. How does this quantification change with similarity and sparsity? Verify that the proposal has accuracy close to vanilla minwise hashing.

3. Verify that there is no impact on running time of the proposed scheme over existing densification schemes, and our proposal is significantly faster than vanilla minwise hashing. Understand how the running time changes with change in sparsity and k?

6.1. Accuracy

For objectives 1 and 2, we selected 9 different word pairs embedding, generated from new20 corpus, with varying level of similarity and sparsity. We use the popular term-document vector representation for each word. The statis-

	Existing h^+			This Paper h^*			Vanilla MinHash		
	100	200	300	100	200	300	100	200	300
RCV1	62	86	104	56	90	116	376	733	1098
URL	373	584	624	287	459	631	2760	6247	8158
NEWS20	154	179	205	139	158	174	1201	2404	3650

Table 2. Time (in milliseconds) requires to compute 100, 200 and 300 hashes of the full data using the existing densification, proposed densification and vanilla minwise hashing. Minhash can be 10-18x slower for computing 300 hashes on these datasets.

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

Table 3. Basic Statistics of Datasets.

tics of these word vector pairs are summarized in Table 1

For each word pairs, we generated k hashes using three different schemes: 1) Densification h, 2) Improved Densification h^+ and the proposed densification h^* (Algorithm 1). Using these hashes, we estimate the Jaccard similarity (Equation 4). We plot the mean square error (MSE) with varying the number of hashes. Since the process is randomized, we repeat the process 5000 times, for every k, and report the average over independent runs. We report all integer values of k in the interval $[1, 2^{14}]$.

It should be noted that since all three schemes have the *LSH Property*, the bias is zero and hence the MSE is the theoretical variance. To validate our variance formula, we also compute and plot the theoretical value of the variance (Equation 12) of the optimal scheme. Also, to understand how all these fast methodologies compare with the accuracy of vanilla minwise hashing we also plot the theoretical variance of minwise hashing which is $\frac{R(1-R)}{k}$.

From the results in Figure 3, we can conclude.
Conclusion 1: The proposed densification is significantly more accurate, irrespective of the choice of sparsity and similarity, than the existing densification schemes especially for large k. Note the y-axis of plots is on log scale, so the accuracy gains are drastic.
Conclusion 2: The gains with optimal densification is more for sparse data.
Conclusion 3: The accuracy of optimal densification is very close the accuracy of costly minwise hashing.
Conclusion 4: The theoretical variance of our proposal overlaps with the empirical estimates, and it seems to go to zero validating Theorem 12.
Conclusion 5: The variances (or the MSE) of existing densification seems to converge to constant and do not go to zero confirming Theorem 1.

6.2. Speed

To compute the runtime, we use three publicly available text datasets: 1) RCV1, 2) URL and 3) News20. The dimensionality and sparsity of these datasets are an excellent representative of the scale and the size frequently encountered in large-scale data processing systems, such as Google's SIBYL (Chandra et al., 2010). The statistics of these datasets are summarized in Table 3.

	100 Bins	200 Bins	300 Bins
RCV1	53	143	238
URL	33	112	202
News20	14	58	120

Table 4. Avg. Number of Empty Bins per vector (rounded) generated with One Permutation Hashing (Li et al., 2012). For sparse datasets, a significant (80% with 300 hashes) of the bins can be empty and have no information. They cannot be used for indexing and kernel learning. Fortunately, we can efficiently densify them with optimal densification, such that, all the generated k hashes have variance similar to minwise hashing. The overall computational cost is significantly less compared to minwise hashing

We implemented three methodologies for computing hashes: 1) Densification Scheme h^+, 2) The Proposed h^* (Algorithm 1 and 3) Vanilla Minwise Hashing. The methods were implemented in C++. Cheap hash function replaced costly permutations. Clever alternatives to avoid mod operations were employed. These tricks ensured that our implementations[2] are as efficient as the possible. We compute the wall clock time required to calculate 100, 200 and 300 hashes of all the three datasets. The time include the end-to-end hash computation of the complete data. Data loading time is not included. The results are presented in Table 2. All the experiments were done on Intel i7-6500U processor laptop with 16GB RAM.

Also, to get an estimate of the importance of densification, we also show the average number of empty bins generated by only using one permutation hashing and report the numbers in Table 4. We can clearly see that the number of empty bins is significantly larger and the hashes are unusable without densification. From Table 2, we conclude:
Conclusion 1: Optimal densification is as fast as traditional densification irrespective of k and the sparsity. However, optimal densification is significantly more accurate.
Conclusion 2: Both the densification scheme is significantly faster than minwise hashing. They are 10-18x faster for computing 300 hashes on the selected datasets.

[2]Codes are available at `http://rush.rice.edu/fastest-minwise.html`

Acknowledgements

This work was supported by NSF IIS-1652131 Grant.

References

Bavarian, Mohammad, Ghazi, Badih, Haramaty, Elad, Kamath, Pritish, Rivest, Ronald L., and Sudan, Madhu. The optimality of correlated sampling. *CoRR*, abs/1612.01041, 2016. URL http://arxiv.org/abs/1612.01041.

Bayardo, Roberto J., Ma, Yiming, and Srikant, Ramakrishnan. Scaling up all pairs similarity search. In *WWW*, pp. 131–140, 2007.

Broder, Andrei Z. On the resemblance and containment of documents. In *the Compression and Complexity of Sequences*, pp. 21–29, Positano, Italy, 1997.

Broder, Andrei Z., Charikar, Moses, Frieze, Alan M., and Mitzenmacher, Michael. Min-wise independent permutations. In *STOC*, pp. 327–336, Dallas, TX, 1998.

Buehrer, Gregory and Chellapilla, Kumar. A scalable pattern mining approach to web graph compression with communities. In *WSDM*, pp. 95–106, Stanford, CA, 2008.

Carter, J. Lawrence and Wegman, Mark N. Universal classes of hash functions. In *STOC*, pp. 106–112, 1977.

Chandra, Tushar, Ie, Eugene, Goldman, Kenneth, Llinares, Tomas Lloret, McFadden, Jim, Pereira, Fernando, Redstone, Joshua, Shaked, Tal, and Singer, Yoram. Sibyl: a system for large scale machine learning. Technical report, 2010.

Charikar, Moses S. Similarity estimation techniques from rounding algorithms. In *STOC*, pp. 380–388, Montreal, Quebec, Canada, 2002.

Chien, Steve and Immorlica, Nicole. Semantic similarity between search engine queries using temporal correlation. In *WWW*, pp. 2–11, 2005.

Chierichetti, Flavio, Kumar, Ravi, Lattanzi, Silvio, Mitzenmacher, Michael, Panconesi, Alessandro, and Raghavan, Prabhakar. On compressing social networks. In *KDD*, pp. 219–228, Paris, France, 2009.

Dahlgaard, Søren, Knudsen, Mathias Bæk Tejs, Rotenberg, Eva, and Thorup, Mikkel. Hashing for statistics over k-partitions. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pp. 1292–1310. IEEE, 2015.

Dahlgaard, Søren, Knudsen, Mathias Bæk Tejs, and Thorup, Mikkel. Fast similarity sketching. *arXiv preprint arXiv:1704.04370*, 2017.

Fetterly, Dennis, Manasse, Mark, Najork, Marc, and Wiener, Janet L. A large-scale study of the evolution of web pages. In *WWW*, pp. 669–678, Budapest, Hungary, 2003.

Haeupler, Bernhard, Manasse, Mark, and Talwar, Kunal. Consistent weighted sampling made fast, small, and easy. *arXiv preprint arXiv:1410.4266*, 2014.

Henzinger, Monika Rauch. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *SIGIR*, pp. 284–291, 2006.

Indyk, Piotr and Motwani, Rajeev. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pp. 604–613, Dallas, TX, 1998.

Li, Ping. 0-bit consistent weighted sampling. In *KDD*, 2015.

Li, Ping, Shrivastava, Anshumali, Moore, Joshua, and König, Arnd Christian. Hashing algorithms for large-scale learning. In *NIPS*, Granada, Spain, 2011.

Li, Ping, Owen, Art B, and Zhang, Cun-Hui. One permutation hashing. In *NIPS*, Lake Tahoe, NV, 2012.

Luo, Chen and Shrivastava, Anshumali. Ssh (sketch, shingle, & hash) for indexing massive-scale time series. In *NIPS 2016 Time Series Workshop*, pp. 38–58, 2017.

Najork, Marc, Gollapudi, Sreenivas, and Panigrahy, Rina. Less is more: sampling the neighborhood graph makes salsa better and faster. In *WSDM*, pp. 242–251, Barcelona, Spain, 2009.

Ondov, Brian D, Treangen, Todd J, Melsted, Páll, Mallonee, Adam B, Bergman, Nicholas H, Koren, Sergey, and Phillippy, Adam M. Mash: fast genome and metagenome distance estimation using minhash. *Genome Biology*, 17(1):132, 2016.

Shrivastava, Anshumali. Simple and efficient weighted minwise hashing. In *Advances in Neural Information Processing Systems*, pp. 1498–1506, 2016.

Shrivastava, Anshumali and Li, Ping. Densifying one permutation hashing via rotation for fast near neighbor search. In *ICML*, Beijing, China, 2014a.

Shrivastava, Anshumali and Li, Ping. In defense of minhash over simhash. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, pp. 886–894, 2014b.

Shrivastava, Anshumali and Li, Ping. Improved densification of one permutation hashing. In *UAI*, Quebec, CA, 2014c.

Weinberger, Kilian, Dasgupta, Anirban, Langford, John, Smola, Alex, and Attenberg, Josh. Feature hashing for large scale multitask learning. In *ICML*, pp. 1113–1120, 2009.