Theory and Applications of b-Bit Minwise Hashing

Ping Li
Department of Statistical Science
Faculty of Computing and Information Science
Cornell University, Ithaca, NY 14853

Arnd Christian König
Microsoft Research
Microsoft Corporation
Redmond, WA 98052

ABSTRACT

Efficient (approximate) computation of set similarity in very large datasets is a common task with many applications in information retrieval and data management. One common approach for this task is *minwise hashing*. This paper describes *b-bit minwise hashing*, which can provide an order of magnitude improvements in storage requirements and computational overhead over the original scheme in practice.

We give both theoretical characterizations of the performance of the new algorithm as well as a practical evaluation on large real-life datasets and show that these match very closely. Moreover, we provide a detailed comparison to other important alternative techniques proposed for estimating set similarities. Our technique yields a very simple algorithm and can be realized with only minor modifications to the original minwise hashing scheme.

1. INTRODUCTION

With the advent of the Internet, many applications are faced with very large and inherently high-dimensional datasets. A common task on these is *similarity search*, i.e., given a high-dimensional data point, the retrieval of data points that are close under a given distance function. In many scenarios, the storage and computational requirements for computing exact distances between all data points are prohibitive, making data representations that allow compact storage and efficient approximate distance computation necessary.

In this paper, we describe *b-bit minwise hashing*, which leverages properties common to many application scenarios to obtain order-of-magnitude improvements in the storage space and computational overhead required for a given level of accuracy over existing techniques. Moreover, while the theoretical analysis of these gains is technically challenging, the resulting algorithm is simple and easy to implement.

To describe our approach, we first consider the concrete task of web page duplicate detection, which is of critical importance in the context of web search and was one of the motivations for the development of the original *minwise hashing* algorithm by Broder et al [2, 4]. Here, the task is to identify pairs of pages that are textually very similar. For this purpose, web pages are modeled as "a set of shingles," where a shingle corresponds to a string of w contiguous words occurring on the page. Now, given two such sets $S_1,\ S_2 \subseteq \Omega, |\Omega| = D$, the normalized similarity known as *resemblance* or *Jaccard similarity*, denoted by R, is

$$R = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = \frac{a}{f_1 + f_2 - a}, \quad \text{where } f_1 = |S_1|,\ f_2 = |S_2|.$$

Duplicate detection now becomes the task of detecting pairs of pages for which R exceeds a threshold value. Here, w is a tuning parameter and was set to be $w = 5$ in several studies [2, 4, 8]. Clearly, the total number of possible shingles is huge. Considering 10^5 unique English words, the total number of possible 5-shingles should be $D = (10^5)^5 = O(10^{25})$. A prior study used $D = 2^{64}$ [8] and even earlier studies used $D = 2^{40}$ [2,4]. Due to the size of D and the number of pages crawled as part of web search, computing the exact similarities for all pairs of pages may require prohibitive storage and computational overhead, leading to approximate techniques based on more compact data structures.

1.1 Minwise Hashing

To address this issue, Broder and his colleagues developed *minwise hashing* in their seminal work [2, 4]. Here, we give a brief introduction to this algorithm. Suppose a random permutation π is performed on Ω, i.e.,

$$\pi:\ \Omega \longrightarrow \Omega, \qquad \text{where} \quad \Omega = \{0, 1, ..., D-1\}.$$

An elementary probability argument shows that

$$\mathbf{Pr}\left(\min(\pi(S_1)) = \min(\pi(S_2))\right) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = R. \tag{1}$$

After k minwise independent permutations, π_1, π_2, ..., π_k, one can estimate R without bias, as a binomial probability:

$$\hat{R}_M = \frac{1}{k}\sum_{j=1}^{k} 1\{\min(\pi_j(S_1)) = \min(\pi_j(S_2))\}, \tag{2}$$

$$\mathrm{Var}\left(\hat{R}_M\right) = \frac{1}{k}R(1-R). \tag{3}$$

We will frequently use the terms "**sample**" and "**sample size**" (i.e., k). For minwise hashing, a sample is a hashed value, $\min(\pi_j(S_i))$, which may require e.g., 64 bits [8].

Since the original minwise hashing work [2, 4], there have been considerable theoretical and methodological developments [3, 5, 13, 15, 17, 18, 22].

Applications: As a general technique for estimating set similarity, minwise hashing has been applied to a wide range of applications, for example, content matching for online advertising [23], detection of redundancy in enterprise file

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 2008 ACM 0001-0782/08/0X00 ...$5.00.

systems [9], syntactic similarity algorithms for enterprise information management [6], Web spam [25], etc.

Many of the applications of minwise hashing are targeted at detecting duplicates or pairs of somewhat high similarity. By proposing an estimator that is particularly accurate for these scenarios, we can reduce the required storage and computational overhead dramatically. Here, the computational savings are a function of how the minwise hashes are used. For any technique that does compute the pairwise similarity for (a large subset of) all pairs, the computation is typically bound by the speed at which the samples can be brought into memory (as the computation itself is simple); hence, the space-reduction our technique offers directly translates into order-of-magnitude speed-up as well.

However, even with the data-size reduction, computing all pairwise similarities is prohibitively expensive in many scenarios. This has lead to a number of approaches that avoid this computation by grouping (subsets of) the samples into buckets and only computing the pairwise similarities for items within the same (set of) buckets. This approach avoids the quadratic number of comparisons, at the cost of some loss in accuracy. Examples of such approaches are the *supershingles* [4] or techniques based on *Locally Sensitive Hashing (LSH)* [1, 5, 14] (also see Chapter 3 of [24] for an excellent detailed explanation of LSH and see [7] for nice applications of LSH ideas in mining associations).

1.2 b-bit Minwise Hashing

In this paper, we establish a unified theoretical framework for ***b-bit minwise hashing***. In our scheme, a **sample** consists of b bits only, as opposed to e.g., $b = 64$ bits [8] in the original minwise hashing. Intuitively, using fewer bits per sample will increase the estimation variance, compared to (3), at the same **sample size** k. Thus, we will have to increase k to maintain the same accuracy. Interestingly, our theoretical results will demonstrate that, when resemblance is not too small (which is the case in many applications, e.g., consider $R \geq 0.5$, the threshold used in [2,4]), we do not have to increase k much. This means our proposed b-bit minwise hashing can be used to improve estimation accuracy and significantly reduce storage requirements at the same time.

For example, when $b = 1$ and $R = 0.5$, the estimation variance will increase at most by a factor of 3. In order not to lose accuracy, we have to increase the sample size by a factor of 3. If we originally stored each hashed value using 64 bits, the improvement by using $b = 1$ will be $64/3 = 21.3$.

Algorithm 1 illustrates the procedure of b-bit minwise hashing, based on the theoretical results in Section 2.

Algorithm 1 The b-bit minwise hashing algorithm, applied to estimating pairwise resemblances in a collection of N sets.

Input: Sets $S_n \subseteq \Omega = \{0, 1, ..., D-1\}$, $n = 1$ to N.
Pre-processing:
1): Generate k random permutations $\pi_j : \Omega \to \Omega$, $j = 1$ to k.
2): For each set S_n and each permutation π_j, store the lowest b bits of $\min(\pi_j(S_n))$, denoted by e_{n,i,π_j}, $i = 1$ to b.
Estimation: (Use two sets S_1 and S_2 as an example.)
1): Compute $\hat{P}_b = \frac{1}{k}\sum_{j=1}^{k}\left\{\prod_{i=1}^{b} 1\{e_{1,i,\pi_j} = e_{2,i,\pi_j}\}\right\}$.
2): Estimate the resemblance by $\hat{R}_b = \frac{\hat{P}_b - C_{1,b}}{1 - C_{2,b}}$, where $C_{1,b}$ and $C_{2,b}$ are from Theorem 1 in Section 2.

1.3 Related Work

Locality Sensitive Hashing (LSH) [1, 5, 14] is a set of techniques for performing approximate search in high dimensions. In the context of estimating set intersections, there exist LSH families for estimating the *resemblance*, the *arccosine* and the *hamming distance*. Our b-bit minwise hashing proposes a new construction of an LSH family (Section 7.4).

In [5, 11], the authors describe hashing schemes that map objects to $\{0, 1\}$. The algorithms for the construction, however, are problem specific. Three discovered 1-bit schemes are (i) the *simhash* [5] based on *sign random projection* [12, 19]; (ii) the hamming distance algorithm based on simple random sampling [14]; and (iii) the hamming distance algorithm based on a variant of random projection [16].

Section 4 will compare our method with two *hamming distance* algorithms [14, 16]. We also wrote a report (`http://www.stat.cornell.edu/~li/b-bit-hashing/RP_minwise.pdf`), which demonstrated that, unless the similarity is very low, b-bit minwise hashing outperforms sign random projections.

A related approach is *Conditional Random Sampling* (CRS) [17, 18] which uses only a single permutation and instead of a single minimum retains as set of the smallest hashed values. CRS provides more accurate (in some scenarios substantially so) estimators for binary data and naturally extends to real-value data and dynamic streaming data; moreover, the same set of hashed values can be used to estimate a variety of summary statistics including histograms, l_p distances (for any p), number of distinct values, χ^2 distances, entropies, etc. However, we have not developed a b-bit scheme for CRS, which appears to be a challenging task.

2. THE FUNDAMENTAL RESULTS

Consider two sets $S_1, S_2 \subseteq \Omega = \{0, 1, 2, ..., D-1\}$, $f_1 = |S_1|$, $f_2 = |S_2|$, $a = |S_1 \cap S_2|$. Apply a random permutation π on S_1 and S_2, where $\pi : \Omega \longrightarrow \Omega$. Define the minimum values under π to be z_1 and z_2:

$$z_1 = \min(\pi(S_1)), \qquad z_2 = \min(\pi(S_2)). \tag{4}$$

Define $e_{1,i} = i$th lowest bit of z_1, and $e_{2,i} = i$th lowest bit of z_2. Theorem 1 derives the main probability formula. Its proof assumes that D is large, which is virtually always satisfied in practice. This result is a good example of approaching a difficult problem by reasonable approximations.

THEOREM 1. *Assume D is large.*

$$P_b = \mathbf{Pr}\left(\prod_{i=1}^{b} 1\{e_{1,i} = e_{2,i}\}\right) = C_{1,b} + (1 - C_{2,b})\, R \tag{5}$$

$$r_1 = \frac{f_1}{D}, \qquad r_2 = \frac{f_2}{D},$$

$$C_{1,b} = A_{1,b}\frac{r_2}{r_1 + r_2} + A_{2,b}\frac{r_1}{r_1 + r_2}, \tag{6}$$

$$C_{2,b} = A_{1,b}\frac{r_1}{r_1 + r_2} + A_{2,b}\frac{r_2}{r_1 + r_2}, \tag{7}$$

$$A_{1,b} = \frac{r_1[1 - r_1]^{2^b - 1}}{1 - [1 - r_1]^{2^b}}, \qquad A_{2,b} = \frac{r_2[1 - r_2]^{2^b - 1}}{1 - [1 - r_2]^{2^b}}. \square \tag{8}$$

The intuition for the difference between (5) and the equivalent equation for minwise hashing (1) is that even when $R = 0$, the collision probability P_b (i.e., the probability that two minima agree on their last b bits) is not zero, but rather $C_{1,b}$. Having to account for this type of "false positives"

makes the derivation more difficult, resulting in the additional terms in (5). Of course, as expected, if $R = 1$, then $P_b = 1$ (because in this case $r_1 = r_2$ and $C_{1,b} = C_{2,b}$).

Note that the only assumption needed in the proof of Theorem 1 is that D is large, which is virtually always satisfied in practice. Interestingly, (5) is remarkably accurate even for very small D. Figure 1 shows that when $D = 20$ ($D = 500$), the absolute error caused by using (5) is < 0.01 (< 0.0004).

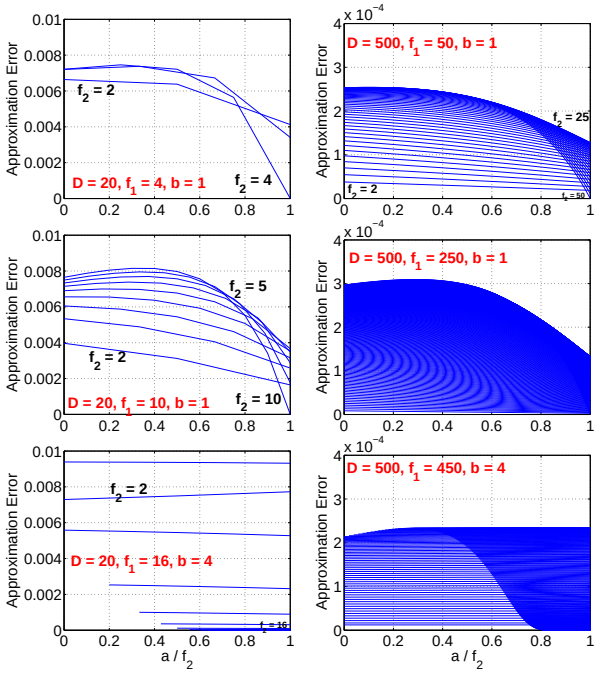

Figure 1: The absolute errors (approximate - exact) by using (5) are very small even for $D = 20$ (left panels) or $D = 500$ (right panels). The exact probability can be numerically computed for small D (from a probability matrix of size $D \times D$). For each D, we selected three f_1 values. We always let $f_2 = 2, 3, ..., f_1$ and $a = 0, 1, 2, ..., f_2$.

2.1 The Unbiased Estimator

Theorem 1 suggests an unbiased estimator $\hat{R}_b$ for R:

$$\hat{R}_b = \frac{\hat{P}_b - C_{1,b}}{1 - C_{2,b}}, \tag{9}$$

$$\hat{P}_b = \frac{1}{k}\sum_{j=1}^{k}\left\{\prod_{i=1}^{b} 1\{e_{1,i,\pi_j} = e_{2,i,\pi_j}\}\right\}, \tag{10}$$

where e_{1,i,π_j} (e_{2,i,π_j}) denotes the ith lowest bit of z_1 (z_2), under the permutation π_j. The variance is,

$$\mathrm{Var}\left(\hat{R}_b\right) = \frac{\mathrm{Var}\left(\hat{P}_b\right)}{\left[1 - C_{2,b}\right]^2} = \frac{1}{k}\frac{P_b(1 - P_b)}{\left[1 - C_{2,b}\right]^2}$$
$$= \frac{1}{k}\frac{\left[C_{1,b} + (1 - C_{2,b})R\right]\left[1 - C_{1,b} - (1 - C_{2,b})R\right]}{\left[1 - C_{2,b}\right]^2} \tag{11}$$

For large b, $\mathrm{Var}\left(\hat{R}_b\right)$ converges to the variance of $\hat{R}_M$, the estimator for the original minwise hashing:

$$\lim_{b\to\infty}\mathrm{Var}\left(\hat{R}_b\right) = \frac{R(1 - R)}{k} = \mathrm{Var}\left(\hat{R}_M\right).$$

In fact, when $b = 64$, $\mathrm{Var}\left(\hat{R}_b\right)$ and $\mathrm{Var}\left(\hat{R}_M\right)$ are numerically indistinguishable for practical purposes.

2.2 The Variance-Space Trade-off

As we decrease b, the space needed for storing each "sample" will be smaller; the estimation variance (11) at the same sample size k, however, will increase. This variance-space trade-off can be precisely quantified by $B(b; R, r_1, r_2)$:

$$B(b; R, r_1, r_2) = b \times \mathrm{Var}\left(\hat{R}_b\right) \times k$$
$$= \frac{b\left[C_{1,b} + (1 - C_{2,b})R\right]\left[1 - C_{1,b} - (1 - C_{2,b})R\right]}{\left[1 - C_{2,b}\right]^2}. \tag{12}$$

Lower $B(b)$ is better. The ratio, $\frac{B(b_1;R,r_1,r_2)}{B(b_2;R,r_1,r_2)}$, measures the improvement of using $b = b_2$ (e.g., $b_2 = 1$) over using $b = b_1$ (e.g., $b_1 = 64$). Some algebra yields the following Lemma.

LEMMA 1. *If $r_1 = r_2$ and $b_1 > b_2$, then*

$$\frac{B(b_1; R, r_1, r_2)}{B(b_2; R, r_1, r_2)} = \frac{b_1}{b_2}\,\frac{A_{1,b_1}(1 - R) + R}{A_{1,b_2}(1 - R) + R}\,\frac{1 - A_{1,b_2}}{1 - A_{1,b_1}}, \tag{13}$$

is a monotonically increasing function of $R \in [0, 1]$. If $R \to 1$ (which implies $r_1, r_2 \to 1$), then

$$\frac{B(b_1; R, r_1, r_2)}{B(b_2; R, r_1, r_2)} \to \frac{b_1}{b_2}. \tag{14}$$

If $r_1 = r_2$, $b_2 = 1$, $b_1 = 64$ (hence we treat $A_{1,b} = 0$), then

$$\frac{B(b_1; R, r_1, r_2)}{B(1; R, r_1, r_2)} = 64\frac{R}{1 + R - r_1}. \quad \square \tag{15}$$

Suppose the original minwise hashing used $b = 64$, then the maximum improvement of b-bit minwise hashing would be 64-fold, attained when $r_1 = r_2 = 1$ and $R = 1$. In the least favorable situation, i.e., $r_1, r_2 \to 0$, the improvement will still be $64\frac{0.5}{1+0.5} = \frac{64}{3} = 21.3$-fold when $R = 0.5$.

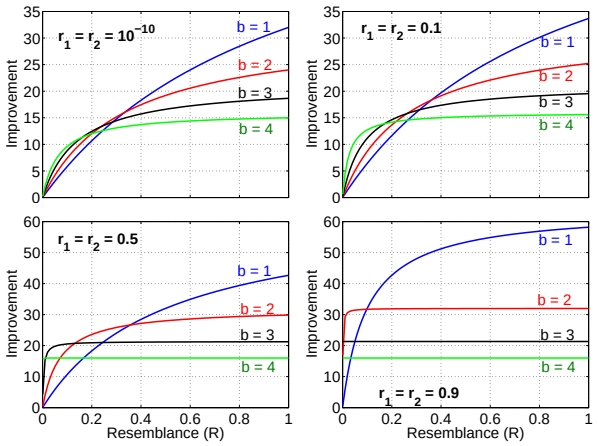

Figure 2: $\frac{B(64)}{B(b)}$, the relative storage improvement of using $b = 1, 2, 3, 4$ bits, compared to using 64 bits. $B(b)$ is defined in (12).

Figure 2 plots $\frac{B(64)}{B(b)}$, to directly visualize the relative improvements, which are consistent with what Lemma 1 predicts. The plots show that, when R is very large (which is the case in many practical applications), it is always good to use $b = 1$. However, when R is small, using larger b may be better. The cut-off point depends on r_1, r_2, R. For example, when $r_1 = r_2$ and both are small, it would be better to use $b = 2$ than $b = 1$ if $R < 0.4$, as shown in Figure 2.

3. EXPERIMENTS

In the following, we evaluate the accuracy of the theoretical derivation and the practical performance of our approach using two sets of experiments. **Experiment 1** is a sanity check, to verify: (i) our proposed estimator $\hat{R}_b$ in (9) is unbiased; (ii) its variance follows the prediction by our formula in (11). **Experiment 2** is a duplicate detection task using a Microsoft proprietary collection of 1,000,000 news articles.

3.1 Experiment 1

The data, extracted from Microsoft Web crawls, consists of 6 pairs of sets. Each set consists of the document IDs which contain the word at least once. We now use b-bit minwise hashing to estimate the similarities of these sets (i.e., we estimate the strength of the word associations).

Table 1: Six word pairs for Experiment 1. For example, "KONG" and "HONG" correspond to two sets of document IDs which contained "KONG" and "HONG" respectively.

Word 1	Word 2	r_1	r_2	R	$\frac{B(64)}{B(1)}$
KONG	HONG	0.0145	0.0143	0.925	31.0
OF	AND	0.570	0.554	0.771	40.8
GAMBIA	KIRIBATI	0.0031	0.0028	0.712	26.6
UNITED	STATES	0.062	0.061	0.591	24.8
LOW	PAY	0.045	0.043	0.112	6.8
A	TEST	0.596	0.035	0.052	6.2

Table 1 summarizes the data and provides the theoretical improvements $\frac{B(64)}{B(1)}$. The words were selected to include highly frequent pairs (e.g., "OF-AND"), highly rare pairs (e.g., "GAMBIA-KIRIBATI"), highly unbalanced pairs (e.g., "A-Test"), highly similar pairs (e.g., "KONG-HONG"), as well as pairs that are not quite similar (e.g., "LOW-PAY").

We estimate the resemblance using the original minwise hashing estimator $\hat{R}_M$ and the b-bit estimator $\hat{R}_b$ ($b = 1, 2, 3$).

Figure 3 plots the empirical mean square errors (MSE = variance + bias^2) in solid lines, and the theoretical variances (11) in dashed lines, for all word pairs. All dashed lines are invisible because they overlap with the corresponding solid curves. Thus, this experiment validates that the variance formula (11) is accurate and $\hat{R}_b$ is indeed unbiased (otherwise, the MSE will differ from the variance).

3.2 Experiment 2: Microsoft News Data

To illustrate the improvements by the use of b-bit minwise hashing on a real-life application, we conducted a duplicate detection experiment using a corpus of 10^6 news documents. The dataset was crawled as part of the BLEWS project at Microsoft [10]. We computed pairwise resemblances for all documents and retrieved documents pairs with resemblance R larger than a threshold $\mathbf{R_0}$. We estimate the resemblances using $\hat{R}_b$ with $b = 1$, 2, 4 bits, and the original minwise hashing. Figure 4 presents the precision & recall curves. The recall values (bottom two panels in Figure 4) are all very high and do not differentiate the estimators.

The precision curves for $\hat{R}_4$ (using 4 bits per sample) and $\hat{R}_M$ (assuming 64 bits per sample) are almost indistinguishable, suggesting a 16-fold improvement in space using $b = 4$.

When using $b = 1$ or 2, the space improvements are normally around 20-fold to 40-fold, compared to $\hat{R}_M$ (assuming 64 bits per sample), especially for achieving high precision.

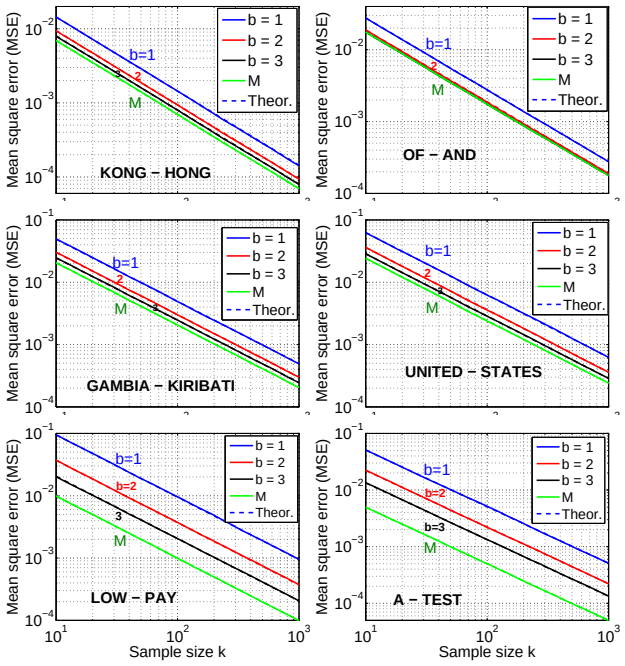

Figure 3: Mean square errors (MSEs). "M" denotes the original minwise hashing. "Theor." denotes the theoretical variances $\mathbf{Var}(\hat{R}_b)$(11) and $\mathbf{Var}(\hat{R}_M)$(3). The dashed curves, however, are invisible because the empirical MSEs overlap the theoretical variances. At the same k, $\mathbf{Var}(\hat{R}_1) > \mathbf{Var}(\hat{R}_2) > \mathbf{Var}(\hat{R}_3) > \mathbf{Var}(\hat{R}_M)$. However, $\hat{R}_1/\hat{R}_2/\hat{R}_3$ only requires 1/2/3 bits per sample, while $\hat{R}_M$ may require 64 bits.

4. COMPARISONS WITH HAMMING DISTANCE ALGORITHMS

Closely related to the resemblance, the *hamming distance* H is another important similarity measure. In the context of hamming distance, a set $S_i \subseteq \Omega = \{0, 1, ..., D-1\}$ is mapped to a D-dimensional binary vector Y_i: $Y_{it} = 1$, if $t \in S_i$ and 0 otherwise. The hamming distance between Y_1 and Y_2 is

$$H = \sum_{i=0}^{D-1} [Y_{1i} \neq Y_{2i}] = |S_1 \cup S_2| - |S_1 \cap S_2| = f_1 + f_2 - 2a,$$

$$i.e., \quad H/D = r_1 + r_2 - 2s.$$

Thus, one can apply b-bit minwise hashing to estimate H, by converting the estimated resemblance $\hat{R}_b$ (9) to $\hat{H}_b$:

$$\hat{H}_b = f_1 + f_2 - 2\frac{\hat{R}_b}{1+\hat{R}_b}(f_1 + f_2) = \frac{1-\hat{R}_b}{1+\hat{R}_b}(f_1 + f_2). \quad (16)$$

The variance of $\hat{H}_b$ can be computed from $\mathrm{Var}\left(\hat{R}_b\right)$ (11) by the "delta method" (i.e., $Var(g(x)) \approx Var(x)\left[g'\left(E(x)\right)\right]^2$):

$$\mathrm{Var}\left(\hat{H}_b\right) = \mathrm{Var}\left(\hat{R}_b\right)\frac{4(r_1+r_2)^2}{(1+R)^4}D^2 + O\left(\frac{1}{k^2}\right). \quad (17)$$

We will first compare $\hat{H}_b$ with an algorithm based on simple random sampling [14] and then with another algorithm based on a variant of random projection [16].

4.1 Simple Random Sampling Algorithm

To reduce the storage, we can randomly sample k coordinates from the original data Y_1 and Y_2 in D-dimensions.

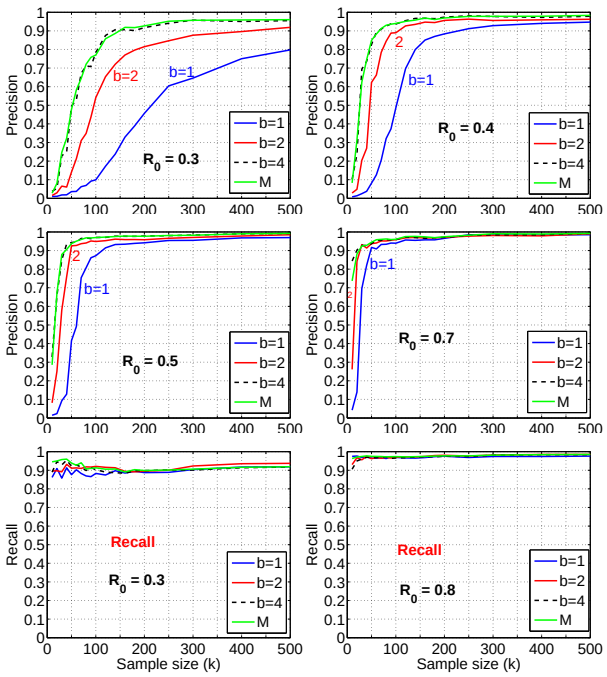

Figure 4: The task is to retrieve news article pairs with resemblance $R \geq R_0$. The recall curves can not differentiate estimators. The precision curves are more interesting. When $R_0 = 0.4$, to achieve a precision = 0.80, the estimators $\hat{R}_M$, $\hat{R}_4$, $\hat{R}_2$, and $\hat{R}_1$ require $k = 50, 50, 75, 145$, respectively, indicating $\hat{R}_4$, $\hat{R}_2$, and $\hat{R}_1$ respectively improve $\hat{R}_M$ (assuming 64 bits per sample) by 16-, 21.4-, and 22-fold. The improvement becomes larger as R_0 increases.

The samples, denoted by h_1 and h_2, are k-dimensional bit vectors, from which we can estimate H:

$$\hat{H}_s = \frac{D}{k}\sum_{j=1}^{k}\left[h_{1j} \neq h_{2j}\right], \tag{18}$$

whose variance would be (assuming $k \ll D$)

$$\mathrm{Var}\left(\hat{H}_s\right) = \frac{D^2}{k}\left[\frac{H}{D} - \frac{H^2}{D^2}\right]. \tag{19}$$

Comparing the two variances, (17) and (19), we find the variance of using simple random sampling, i.e., $Var\left(\hat{H}_s\right)$, is substantially larger than the variance of using b-bit minwise hashing, i.e., $Var\left(\hat{H}_b\right)$, especially when the data are sparse. We consider in practice one will most likely implement the random sampling algorithm by storing only the original locations (coordinates) of the non-zeros in the samples. If we do so, the total bits on average will be $\frac{r_1+r_2}{2}64k$ (per set). This motivates us to define the following ratio:

$$G_{s,b} = \frac{\mathrm{Var}\left(\hat{H}_s\right) \times \frac{r_1+r_2}{2}64k}{\mathrm{Var}\left(\hat{H}_b\right) \times bk}. \tag{20}$$

to compare the storage costs. Recall each sample of b-bit minwise hashing requires b bits (i.e., bk bits per set). The following Lemma may help characterize the improvement:

LEMMA 2. *If $r_1, r_2 \to 0$, then $G_{s,b}$ as defined in (20)*

$$G_{s,b} \to \frac{8}{b}\frac{(2^b-1)(1+R)^3}{1+(2^b-1)R}. \quad \square \tag{21}$$

In other words, for small r_1, r_2, $G_{s,b} \approx \frac{8}{b}\left(2^b-1\right)$ if $R \approx 0$; and $G_{s,b} \approx \frac{64}{b}\frac{2^b-1}{2^b}$, if $R \approx 1$. Figure 5 plots $G_{s,b=1}$, verifying the substantial improvement of b-bit minwise hashing over simple random sampling (often 10 to 30-fold).

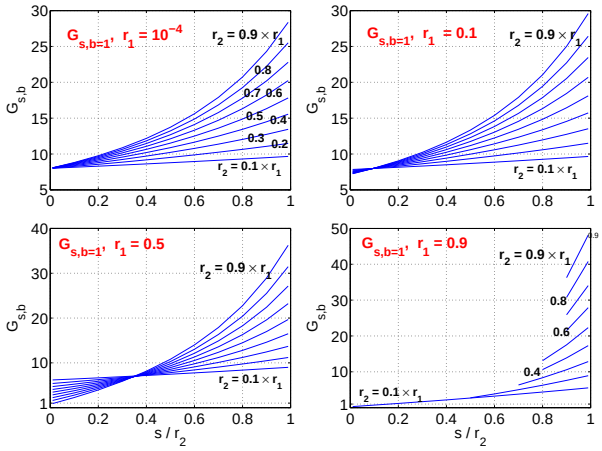

Figure 5: $G_{s,b=1}$ as defined in (20) for illustrating the improvement of b-bit minwise hashing over simple random sampling. We consider $r_1 = 10^{-4}, 0.1, 0.5, 0.9$, r_2 ranging from $0.1r_1$ to $0.9r_1$, and s from 0 to r_2. Note that $r_1 + r_2 - s \leq 1$ has to be satisfied.

4.2 Random Projection + Modular Arithmetic

An interesting 1-bit scheme was developed in [16] using random projection followed by modular arithmetic. A random matrix $\mathbf{U} \in \mathbb{R}^{D\times k}$ is generated with entries being i.i.d. samples u_{ij} from a binomial distribution: $u_{ij} = 1$ with probability $\frac{\beta}{2}$ and $u_{ij} = 0$ with probability $1-\frac{\beta}{2}$. Let $v_1 = Y_1 \times U \pmod 2$ and $v_2 = Y_2 \times U \pmod 2$. [16] showed that

$$E_\beta = \mathbf{Pr}\left(v_{1,j} \neq v_{2,j}\right) = \frac{1}{2}\left(1 - [1-\beta]^H\right) \tag{22}$$

which allows us to estimate the hamming distance H by

$$\hat{H}_{rp,\beta} = \frac{\log(1-2\hat{E}_\beta)}{\log(1-\beta)} \tag{23}$$

We calculate the variance of $\hat{H}_{rp,\beta}$ to be

$$Var\left(\hat{H}_{rp,\beta}\right) = \frac{1}{k}\frac{4E_\beta(1-E_\beta)}{(1-2E_\beta)^2\log^2(1-\beta)} + O\left(\frac{1}{k^2}\right) \tag{24}$$

which suggests that the performance of this 1-bit scheme might be sensitive to β, which must be pre-determined for all sets at the processing time (i.e., it can not be modified in the estimation phrase for a particular pair). Figure 6 provides the "optimal" β (denoted by β^*) values (as function of H) by numerically minimizing the variance (24).

It is interesting to compare this random projection-based 1-bit scheme with our b-bit minwise hashing using the following ratio of their variances:

$$G_{rp,b,\beta} = \frac{\mathrm{Var}\left(\hat{H}_{rp,\beta}\right) \times k}{\mathrm{Var}\left(\hat{H}_b\right) \times bk}. \tag{25}$$

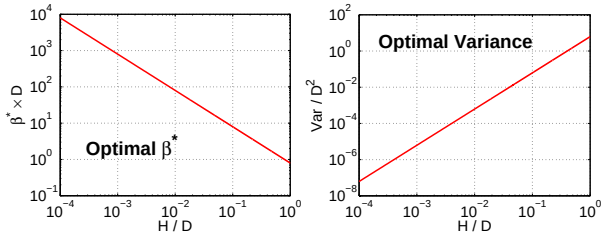

Figure 6: Left panel: the β^* values at which the smallest variances (24) are attained. Right panel: the corresponding optimal variances.

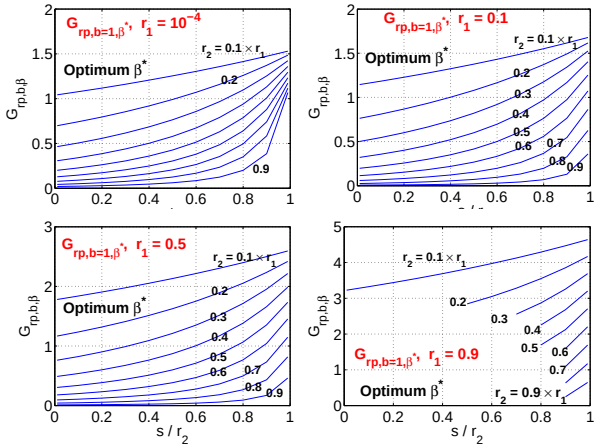

Figure 7: $G_{rp,b=1,\beta^*}$ as defined in (25) for comparing $\hat{H}_b$ with $\hat{H}_{rp,\beta^*}$. For each combination of r_1, r_2, s, we computed and used the optimal β^* for the variance.

Figure 7 shows that if it is possible to choose the optimal β^* for random projection, one can achieve good performance, similar to (or even better than) b-bit minwise hashing.

The problem is that we must choose the same β for all sets. Figure 8 presents a typical example, which uses $H^*/D = 10^{-4}$ to compute the "optimal" β for a wide range of (r_1, r_2, s) values. The left bottom panel illustrates that when $r_1 = 10^{-4}$ using this particular choice of β results in fairly good performance compared to b-bit minwise hashing. (Recall $H/D = r_1 + r_2 - 2s$.) As soon as the true H substantially deviates from the guessed H^*, the performance of $\hat{H}_{rp,\beta}$ using random projection degrades dramatically.

There is one more issue. At the optimal $\beta^*(H)$, our calculations show that the probability (22) $E_{\beta^*} \approx 0.2746$. However, if the chosen $\beta > \beta^*(H)$, then E_β may approach 1/2. As $\hat{E}_\beta$ is random, it is likely that the observed $\hat{E}_\beta > 1/2$, i.e., $\log\left(1 - 2\hat{E}_\beta\right)$ becomes undefined in (23). Thus, it is safer to "over-estimate" H when choosing β. When we have a large collection of sets, this basically means the chosen β will be very different from its optimal value for most pairs.

Finally, Figure 9 provides an empirical study as a sanity check that the variance formula (24) is indeed accurate and that, if the guessed H for selecting β deviates from the true H, then the random projection estimator $\hat{H}_{rp,\beta}$ exhibits much larger errors than the b-bit hashing estimator $\hat{H}_b$.

5. IMPROVEMENT BY COMBINING BITS

Our theoretical and empirical results have confirmed that,

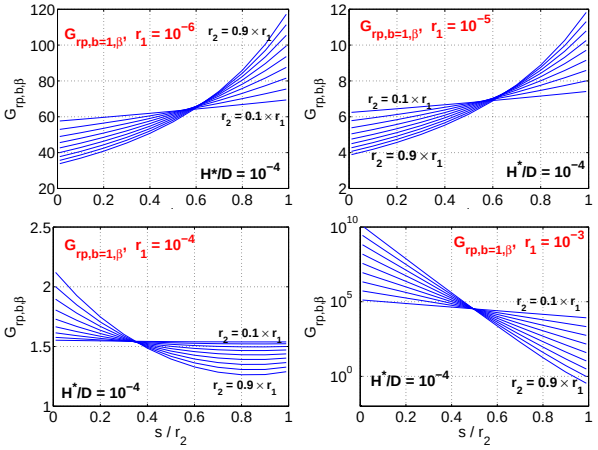

Figure 8: $G_{rp,b=1,\beta}$ (25) computed by using the fixed β which is the optimal β when $H = H^* = 10^{-4}D$.

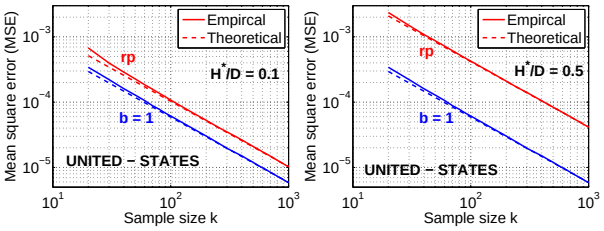

Figure 9: The exact H for this pair of sets is 0.0316. We use the optimal β at $H^*/D = 0.1$ (left panel) and $H^*/D = 0.5$ (right panel). Compared to $\hat{H}_{b=1}$, the MSEs of $\hat{H}_{rp,\beta}$ (labeled by "rp") are substantially larger. The theoretical variances (dashed lines) (24) and (17), essentially overlap the empirical MSEs.

when the resemblance R is reasonably high, even a single bit per sample may contain sufficient information for accurately estimating the similarity. This naturally leads to the conjecture that, when R is close to 1, one might further improve the performance by looking at a combination of multiple bits (i.e., "$b < 1$"). One simple approach is to combine two bits from two permutations using XOR ($\oplus$) operations.

Recall $e_{1,1,\pi}$ denotes the lowest bit of the hashed value under π. Theorem 1 has proved that

$$P_1 = \mathbf{Pr}\left(e_{1,1,\pi} = e_{2,1,\pi}\right) = C_{1,1} + (1 - C_{2,1})\, R$$

Consider two permutations π_1 and π_2. We store

$$x_1 = e_{1,1,\pi_1} \oplus e_{1,1,\pi_2}, \qquad x_2 = e_{2,1,\pi_1} \oplus e_{2,1,\pi_2}$$

Then $x_1 = x_2$ either when $e_{1,1,\pi_1} = e_{2,1,\pi_1}$ and $e_{1,1,\pi_2} = e_{2,1,\pi_2}$, or, when $e_{1,1,\pi_1} \neq e_{2,1,\pi_1}$ and $e_{1,1,\pi_2} \neq e_{2,1,\pi_2}$. Thus

$$T = \mathbf{Pr}\left(x_1 = x_2\right) = P_1^2 + (1 - P_1)^2, \tag{26}$$

which is a quadratic equation with a solution:

$$\hat{R}_{1/2} = \frac{\sqrt{\max\{2\hat{T} - 1, 0\}} + 1 - 2C_{1,1}}{2 - 2C_{2,1}}. \tag{27}$$

This estimator is slightly biased at small sample size k. We use $\hat{R}_{1/2}$ to indicate that two bits are combined into one (but each sample is still stored using 1 bit). The asymptotic variance of $\hat{R}_{1/2}$ can be derived to be

$$\mathrm{Var}\left(\hat{R}_{1/2}\right) = \frac{1}{k}\frac{T(1-T)}{4(1-C_{2,1})^2(2T-1)} + O\left(\frac{1}{k^2}\right). \tag{28}$$

Interestingly, as $R \to 1$, $\hat{R}_{1/2}$ does twice as well as $\hat{R}_1$:

$$\lim_{R\to 1} \frac{\mathrm{Var}\left(\hat{R}_1\right)}{\mathrm{Var}\left(\hat{R}_{1/2}\right)} = \lim_{R\to 1} \frac{2(1-2P_1)^2}{(1-P_1)^2+P_1^2} = 2. \qquad (29)$$

On the other hand, $\hat{R}_{1/2}$ may not be good when R is not too large. For example, one can numerically show that

$$\mathrm{Var}\left(\hat{R}_1\right) < \mathrm{Var}\left(\hat{R}_{1/2}\right), \quad \text{if } R < 0.5774,\ r_1,\ r_2 \to 0$$

Figure 10 plots the empirical MSEs for two word pairs in Experiment 1, for $\hat{R}_{1/2}$, $\hat{R}_1$, and $\hat{R}_M$. For the highly similar pair, "KONG-HONG," $\hat{R}_{1/2}$ exhibits superior performance compared to $\hat{R}_1$. For "UNITED-STATES," whose $R = 0.591$, $\hat{R}_{1/2}$ performs similarly to $\hat{R}_1$.

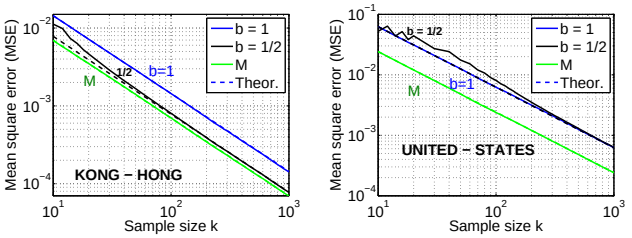

Figure 10: MSEs for comparing $\hat{R}_{1/2}$ (27) with $\hat{R}_1$ and $\hat{R}_M$. Due to the bias of $\hat{R}_{1/2}$, the theoretical variances Var $\left(\hat{R}_{1/2}\right)$, i.e., (28), deviate from the empirical MSEs when k is small.

In summary, for applications which care about very high similarities, combining bits can reduce storage even further.

6. COMPUTATIONAL IMPROVEMENTS

When computing set similarity for large sets of samples, the key operation is determining the number of identical b-bit samples. While samples for values of b that are multiples of 16 bits can easily be compared using a single machine instruction, efficiently computing the overlap between b-bit samples for small b is less straightforward. In the following, we will describe techniques for computing the number of identical b-bit samples when these are packed into arrays $A_l[1, \ldots, \frac{k \cdot b}{w}]$, $l = 1, 2$ of w-bit words. To compute the number of identical b-bit samples, we iterate through the arrays; for each offset h, we first compute $v = A_1[h] \oplus A_2[h]$. Now, the number of b-bit blocks in v that contain only 0s corresponds to the number of identical b-bit samples.

The case of $b = 1$ corresponds to the problem of counting the number of 0-bits in a word. We tested a number of different methods and found the fastest approach to be pre-computing an array $bits[1, \ldots, 2^{16}]$ such that $bits[t]$ corresponds to the number of 0-bits in the binary representation of t and using lookups into this array. This approach extends to $b > 1$ as well.

To evaluate this approach we timed a tight loop computing the number of identical samples in two arrays of b-bit hashes covering a total of 1.8 billion 32-bit words (using a 64-bit Intel 6600 Processor). Here, the 1-bit hashing requires 1.67x the time that the 32-bit minwise hashing requires (1.73x when comparing to 64-bit minwise hashing). The results were essentially identical for $b = 2, 4, 8$. Given that, when $R > 0.5$, we can gain a storage reduction of 21.3-fold, we expect the resulting improvement in computational efficiency to be 21.3/1.67 = 12.8-fold in the above setup.

7. EXTENSIONS AND APPLICATIONS

7.1 Three-Way Resemblance

Many applications in data mining or data cleaning require not only estimates of 2-way, but also of multi-way similarities. The original minwise hashing naturally extends to multi-way resemblance. In [21], we extended b-bit minwise hashing to estimate 3-way resemblance. We developed a highly accurate, but complicated estimator, as well as a much simplified estimator suitable for sparse data. Interestingly, at least $b \geq 2$ bits are needed in order to estimate 3-way resemblance. Similar to the 2-way case, b-bit minwise hashing can result in an order-of-magnitude reduction in the storage space required for a given estimation accuracy when testing for moderate to high similarity.

7.2 Large-Scale Machine Learning

A different category of applications for b-bit minwise hashing is machine learning on very large datasets. For example, one of our projects [20] focuses on linear Support Vector Machines (SVM). We were able to show that the resemblance matrix, the minwise hashing matrix, and the b-bit minwise hashing matrix are all positive definite matrices (kernels); and we integrated b-bit minwise hashing with linear SVM. This allows us to significantly speed up training and testing times with almost no loss in classification accuracy for many practical scenarios. In addition, this provides an elegant solution to the problem of SVM training in scenarios where the training data cannot fit in memory.

Interestingly, the technique we used for linear SVM essentially provides a universal strategy for integrating b-bit minwise hashing with many other learning algorithms, for example, logistic regression.

7.3 Improving Estimates by Maximum Likelihood Estimators (MLE)

While b-bit minwise hashing is particularly effective in applications which mainly concern sets of high similarities (e.g., $R > 0.5$), there are other important applications in which not just pairs of high similarities matter. For example, many learning algorithms require all pairwise similarities and it is expected that only a small fraction of the pairs are similar. Furthermore, many applications care more about *containment* (e.g., which fraction of one set is contained in another set) than the *resemblance*. In a recent technical report (`http://www.stat.cornell.edu/~li/b-bit-hashing/AccurateHashing.pdf`), we showed that the estimators for minwise hashing and b-bit minwise hashing used in the current practice can be systematically improved and the improvements are most significant for set pairs of low resemblance and high containment.

For minwise hashing, instead of only using $\mathbf{Pr}(z_1 = z_2)$, where z_1 and z_2 are two hashed values, we can combine it with $\mathbf{Pr}(z_1 < z_2)$ and $\mathbf{Pr}(z_1 > z_2)$ to form a 3-cell multinomial estimation problem, whose maximum likelihood estimator (MLE) is the solution to a cubic equation. For b-bit minwise hashing, we formulate a $2^b \times 2^b$-cell multinomial problem, whose MLE requires a simple numerical procedure.

7.4 The New LSH Family

Applications such as near neighbor search, similarity clustering, and data mining will significantly benefit from b-bit minwise hashing. It is clear that b-bit minwise hashing will significantly improve the efficiency of simple linear algorithms (for near neighbor search) or simple quadratic algorithms (for similarity clustering), when the key bottleneck is main-memory throughput.

Techniques based on *Locality Sensitive Hashing (LSH)* [1, 5, 14] have been successfully used to achieve sub-linear (for near neighbor search) or sub-quadratic (for similarity clustering) performance. It is interesting that b-bit minwise hashing is a new family of LSH and hence in this section we would like to provide more theoretical properties in the context of LSH and approximate near neighbor search.

Consider a set S_1. Suppose there exists another set S_2 whose resemblance distance $(1-R)$ from S_1 is at most d_0, i.e., $1-R \leq d_0$. The goal of *c-approximate d_0-near neighbor* algorithms is to return sets (with high probability) whose resemblance distances from S_1 are at most $c \times d_0$ with $c > 1$.

Recall z_1 and z_2 denote the minwise hashed values for sets S_1 and S_2, respectively. The performance of the LSH algorithm depends on the difference (gap) between the following $P^{(1)}$ and $P^{(2)}$ (respectively corresponding to d_0 and cd_0):

$$\text{If } 1-R \leq d_0, \text{ then } R = \mathbf{Pr}\left(z_1 = z_2\right) \geq 1 - d_0 = P^{(1)}.$$

$$\text{If } 1-R \geq cd_0, \text{ then } R = \mathbf{Pr}\left(z_1 = z_2\right) \geq 1 - cd_0 = P^{(2)}.$$

A larger gap between $P^{(1)}$ and $P^{(2)}$ implies a more efficient LSH algorithm. The following "ρ" value (ρ_M for minwise hashing) characterizes the gap:

$$\rho_M = \frac{\log 1/P^{(1)}}{\log 1/P^{(2)}} = \frac{\log(1-d_0)}{\log(1-cd_0)} \tag{30}$$

A smaller ρ (i.e., larger difference between $P^{(1)}$ and $P^{(2)}$) leads to a more efficient LSH algorithm and $\rho < \frac{1}{c}$ is particularly desirable [1, 14]. The general LSH theoretical result tells us that the query time for c-approximate d_0-near neighbor is dominated by $O(N^\rho)$ distance evaluations, where N is the total number of sets in the collection.

Recall P_b, as defined in (5), denotes the collision probability for b-bit minwise hashing. The ρ_b value for c-approximate d_0-near neighbor search can be computed as follows.

$$1-R \leq d_0 \Longrightarrow P_b \geq C_{1,b} + (1-C_{2,b})(1-d_0)$$
$$1-R \geq cd_0 \Longrightarrow P_b \leq C_{1,b} + (1-C_{2,b})(1-cd_0)$$
$$\rho_b = \frac{\log\left(C_{1,b} + (1-C_{2,b})(1-d_0)\right)}{\log\left(C_{1,b} + (1-C_{2,b})(1-cd_0)\right)} \tag{31}$$

Figure 11 suggests that b-bit minwise hashing can potentially achieve very similar ρ values compared to the original minwise hashing, when the applications care mostly about highly similar sets (e.g., $d_0 = 0.1$, the top panels of Figure 11), even using merely $b = 1$. If the applications concern sets which are not necessarily highly similar (e.g., $d_0 = 0.5$, the bottom panels), using $b = 3$ or 4 will still have similar ρ values as using the original minwise hashing.

We expect that these theoretical properties regarding the ρ values will potentially be useful in future work. We are currently developing new variants of LSH algorithms for near neighbor search based on b-bit minwise hashing.

Subsequent documents will be made available at `www.stat.cornell.edu/~li/b-bit-hashing`, which is a repository for maintaining the papers and technical reports related to b-bit minwise hashing.

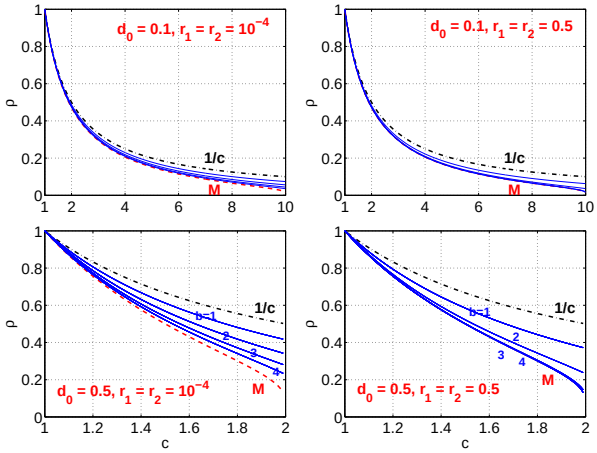

Figure 11: ρ_M and ρ_b defined in (30) and (31) for measuring the potential performance of LSH algorithms. "M" denotes the original minwise hashing.

8. CONCLUSION

Minwise hashing is a standard technique for efficiently estimating set similarity in massive datasets. In this paper, we gave an overview of ***b-bit minwise hashing***, which modifies the original scheme by storing the lowest b bits of each hashed value. We proved that, when the similarity is reasonably high (e.g., resemblance ≥ 0.5), using $b = 1$ bit per hashed value can, even in the worst case, gain a 21.3-fold improvement in storage space (at similar estimation accuracy), compared to storing each hashed value using 64 bits. As many applications are primarily interested in identifying duplicates of reasonably similar sets, these improvements can result in substantial reduction in storage (and consequently computational) overhead in practice.

We also compared our scheme to other approaches that map the hashed objects to single bits, both in theory as well as experimentally.

Our proposed method is simple and requires only minimal modification to the original minwise hashing algorithm. It can be used in the context of a number of different applications, such as duplicate detection, clustering, similarity search and machine learning and we expect that it will be adopted in practice.

ACKNOWLEDGEMENT

This work is supported by NSF (DMS-0808864), ONR (YIP-N000140910911), and a grant from Microsoft. We thank the Board Members for suggesting a direct comparison with [16].

9. REFERENCES

[1] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Commun. ACM*, volume 51, pages 117–122, 2008.

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

[3] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer Systems and Sciences*, 60(3):630–659, 2000.

[4] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. In *WWW*, pages 1157 – 1166, Santa Clara, CA, 1997.

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

[6] Ludmila Cherkasova, Kave Eshghi, Charles B. Morrey III, Joseph Tucek, and Alistair C. Veitch. Applying syntactic similarity algorithms for enterprise information management. In *KDD*, pages 1087–1096, Paris, France, 2009.

[7] Edith Cohen, Mayur Datar, Shinji Fujiwara, Aristides Gionis, Piotr Indyk, Rajeev Motwani, Jeffrey D. Ullman, and Cheng Yang. Finding interesting associations without support pruning. *IEEE Trans. on Knowl. and Data Eng.*, 13(1), 2001.

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

[9] George Forman, Kave Eshghi, and Jaap Suermondt. Efficient detection of large-scale redundancy in enterprise file systems. *SIGOPS Oper. Syst. Rev.*, 43(1):84–91, 2009.

[10] Michael Gamon, Sumit Basu, Dmitriy Belenko, Danyel Fisher, Matthew Hurst, and Arnd Christian König. Blews: Using blogs to provide context for news articles. In *AAAI Conference on Weblogs and Social Media*, 2008.

[11] Aristides Gionis, Dimitrios Gunopulos, and Nick Koudas. Efficient and tunable similar set retrieval. In *SIGMOD*, pages 247–258, Santa Barbara, CA, 2001.

[12] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of ACM*, 42(6):1115–1145, 1995.

[13] Piotr Indyk. A small approximately min-wise independent family of hash functions. *Journal of Algorithms*, 38(1):84–90, 2001.

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

[15] Toshiya Itoh, Yoshinori Takei, and Jun Tarui. On the sample size of k-restricted min-wise independent permutations and other k-wise distributions. In *STOC*, pages 710–718, San Diego, CA, 2003.

[16] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *STOC*, pages 614–623, Dallas, TX, 1998.

[17] Ping Li and Kenneth W. Church. A sketch algorithm for estimating two-way and multi-way associations. *Computational Linguistics (Preliminary results appeared in HLT/EMNLP 2005)*, 33(3):305–354, 2007.

[18] Ping Li, Kenneth W. Church, and Trevor J. Hastie. One sketch for all: Theory and applications of conditional random sampling. In *NIPS (Preliminary results appeared in NIPS 2006)*, Vancouver, BC, Canada, 2008.

[19] Ping Li, Trevor J. Hastie, and Kenneth W. Church. Improving random projections using marginal information. In *COLT*, pages 635–649, Pittsburgh, PA, 2006.

[20] Ping Li, Joshua Moore, and Arnd Christian König. b-bit minwise hashing for large-scale linear SVM. Technical report.

[21] Ping Li, Arnd Christian König, and Wenhao Gui. b-bit minwise hashing for estimating three-way similarities. In *NIPS*, Vancouver, BC, 2010.

[22] Mark Manasse, Frank McSherry, and Kunal Talwar. Consistent weighted sampling. Technical Report MSR-TR-2010-73, Microsoft Research, 2010.

[23] Sandeep Pandey, Andrei Broder, Flavio Chierichetti, Vanja Josifovski, Ravi Kumar, and Sergei Vassilvitskii. Nearest-neighbor caching for content-match applications. In *WWW*, pages 441–450, Madrid, Spain, 2009.

[24] Anand Rajaraman and Jeffrey Ullman. *Mining of Massive Datasets*. http://i.stanford.edu/ ullman/mmds.html.

[25] Tanguy Urvoy, Emmanuel Chauveau, Pascal Filoche, and Thomas Lavergne. Tracking web spam with html style similarities. *ACM Trans. Web*, 2(1):1–28, 2008.