b-Bit Minwise Hashing

Ping Li[*]
Department of Statistical Science
Faculty of Computing and Information Science
Cornell University, Ithaca, NY 14853
pingli@cornell.edu

Arnd Christian König
Microsoft Research
Microsoft Corporation
Redmond, WA 98052
chrisko@microsoft.com

ABSTRACT

This paper establishes the theoretical framework of ***b*-bit minwise hashing**. The original *minwise hashing* method has become a standard technique for estimating set similarity (e.g., *resemblance*) with applications in information retrieval, data management, computational advertising, etc.

By only storing b bits of each hashed value (e.g., $b = 1$ or 2), we gain substantial advantages in terms of storage space. We prove the basic theoretical results and provide an unbiased estimator of the resemblance for any b. We demonstrate that, even in the least favorable scenario, using $b = 1$ may reduce the storage space at least by a factor of 21.3 (or 10.7) compared to $b = 64$ (or $b = 32$), if one is interested in resemblance ≥ 0.5.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data Mining

General Terms

Algorithms, Performance, Theory

1. INTRODUCTION

Computing the size of set intersections is a fundamental problem in information retrieval, databases, and machine learning. Given two sets, S_1 and S_2, where

$$S_1,\ S_2 \subseteq \Omega = \{0, 1, 2, ..., D-1\},$$

a basic task is to compute the joint size $a = |S_1 \cap S_2|$, which measures the (un-normalized) similarity between S_1 and S_2. The ***resemblance***, denoted by R, is a normalized similarity measure:

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

In large datasets encountered in information retrieval and databases, efficiently computing the joint sizes is often highly challenging [3,18]. Detecting duplicate web pages is a classical example [4,6].

Typically, each Web document can be processed as "a bag of shingles," where a shingle consists of w contiguous words in a document. Here w is a tuning parameter and was set to be $w = 5$ in several studies [4, 6, 12]. Clearly, the total number of possible shingles is huge. Considering merely 10^5 unique English words, the total number of possible 5-shingles should be $D = (10^5)^5 = O(10^{25})$. Prior studies used $D = 2^{64}$ [12] and $D = 2^{40}$ [4,6].

1.1 Minwise Hashing

In their seminal work, Broder and his colleagues developed *minwise hashing* and successfully applied the technique to duplicate Web page removal [4,6]. Since then, there have been considerable theoretical and methodological developments [5,8,19,21–23,26].

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 [30], detection of large-scale redundancy in enterprise file systems [14], syntactic similarity algorithms for enterprise information management [27], compressing social networks [9], advertising diversification [17], community extraction and classification in the Web graph [11], graph sampling [29], wireless sensor networks [25], Web spam [24,33], Web graph compression [7], and text reuse in the Web [2].

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:

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

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

Throughout the paper, we frequently use the terms "**sample**" and "**sample size**" (i.e., k). In *minwise hashing*, a sample is a hashed value, $\min(\pi_j(S_i))$, which may require e.g., 64 bits to store [12].

1.2 Our Main Contributions

In this paper, we establish a unified theoretical framework for ***b*-bit minwise hashing**. Instead of using $b = 64$ bits [12] or 40 bits [4, 6], our theoretical results suggest using as few as $b = 1$ or $b = 2$ bits can yield significant improvements.

In b-bit minwise hashing, a **sample** consists of b bits only, as opposed to e.g., 64 bits 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 (e.g., $R \geq 0.5$, the threshold used in [4,6]), 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 this case, in order not to lose accuracy, we have to increase the sample size by a factor of

[*] Supported by Microsoft, NSF-DMS and ONR-YIP.

WWW 2010, April 26–30, 2010, Raleigh, North Carolina, USA.
ACM 978-1-60558-799-8/10/04.

3. If we originally stored each hashed value using 64 bits [12], 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 Sec. 2.

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

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

1.3 Comparisons with LSH Algorithms

Locality Sensitive Hashing (LSH) [8,20] 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* [1].

In [8, 16], the authors describe LSH hashing schemes that map objects to $\{0, 1\}$ (i.e., 1-bit schemes). The algorithms for the construction, however, are problem specific. Two discovered 1-bit schemes are the *sign random projections* (also known as *simhash*) [8] and the *Hamming distance LSH* algorithm proposed by [20].

Our b-bit minwise hashing proposes a new construction, which maps objects to $\{0, 1, ..., 2^b - 1\}$ instead of just $\{0, 1\}$. While our major focus is to compare with the original minwise hashing, we also conduct comparisons with the other two known 1-bit schemes.

1.3.1 Sign Random Projections

The method of sign (1-bit) random projections estimates the *arccosine*, which is $\cos^{-1}\left(\frac{a}{\sqrt{f_1 f_2}}\right)$, using our notation for sets S_1 and S_2. A separate technical report is devoted to comparing b-bit minwise hashing with sign (1-bit) random projections. See `www.stat.cornell.edu/~li/hashing/RP_minwise.pdf`. That report demonstrates that, unless the similarity level is very low, b-bit minwise hashing outperforms sign random projections.

The method of sign random projections has received significant attention in the context of duplicate detection. According to [28], a great advantage of *simhash* over *minwise hashing* is the smaller size of the fingerprints required for duplicate detection. The space-reduction of b-bit *minwise hashing* overcomes this issue.

1.3.2 The Hamming Distance LSH Algorithm

Sec. 4 will compare b-bit minwise hashing with the *Hamming distance LSH* algorithm developed in [20] (and surveyed in [1]):

- When the *Hamming distance LSH* algorithm is implemented naively, to achieve the same level of accuracy, its required storage space will be many magnitudes larger than that of b-bit minwise hashing in sparse data (i.e., $|S_i|/D$ is small).
- If we only store the non-zero locations in the *Hamming distance LSH* algorithm, then its required storage space will be about one magnitude larger (e.g., 10 to 30 times).

2. THE FUNDAMENTAL RESULTS

Consider two sets, S_1 and S_2,

$$S_1, S_2 \subseteq \Omega = \{0, 1, 2, ..., D-1\},$$
$$f_1 = |S_1|,\ f_2 = |S_2|,\ a = |S_1 \cap S_2|$$

Apply a random permutation π on S_1 and S_2: $\pi : \Omega \longrightarrow \Omega$. Define the minimum values under π to be z_1 and z_2:

$$z_1 = \min(\pi(S_1)), \qquad z_2 = \min(\pi(S_2)).$$

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.

THEOREM 1. *Assume D is large.*

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

where (4)

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

$$C_{1,b} = A_{1,b}\frac{r_2}{r_1 + r_2} + A_{2,b}\frac{r_1}{r_1 + r_2}, \tag{5}$$

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

$$A_{1,b} = \frac{r_1[1 - r_1]^{2^b - 1}}{1 - [1 - r_1]^{2^b}}, \qquad A_{2,b} = \frac{r_2[1 - r_2]^{2^b - 1}}{1 - [1 - r_2]^{2^b}}. \tag{7}$$

For a fixed r_j (where $j \in \{1, 2\}$), $A_{j,b}$ is a monotonically decreasing function of $b = 1, 2, 3, ...$

For a fixed b, $A_{j,b}$ is a monotonically decreasing function of $r_j \in$ [0, 1], reaching a limit:

$$\lim_{r_j \to 0} A_{j,b} = \frac{1}{2^b}. \tag{8}$$

***Proof**: See Appendix A.*□

Theorem 1 says that, for a given b, the desired probability (4) is determined by R and the ratios, $r_1 = \frac{f_1}{D}$ and $r_2 = \frac{f_2}{D}$. The only assumption needed in the proof of Theorem 1 is that D should be large, which is always satisfied in practice.

$A_{j,b}$ ($j \in \{1, 2\}$) is a decreasing function of r_j and $A_{j,b} \leq \frac{1}{2^b}$. As b increases, $A_{j,b}$ converges to zero very quickly. In fact, when $b \geq 32$, one can essentially view $A_{j,b} = 0$.

2.1 An Intuitive (Heuristic) Explanation

A simple heuristic argument may provide a more intuitive explanation of Theorem 1. Consider $b = 1$. One might expect that

$$\begin{aligned}\mathbf{Pr}(e_{1,1} = e_{2,1}) =& \mathbf{Pr}(e_{1,1} = e_{2,1}|z_1 = z_2)\mathbf{Pr}(z_1 = z_2)\\ &+ \mathbf{Pr}(e_{1,1} = e_{2,1}|z_1 \neq z_2)\mathbf{Pr}(z_1 \neq z_2)\\ &\overset{??}{\approx} R + \frac{1}{2}(1 - R) = \frac{1 + R}{2},\end{aligned}$$

because when z_1 and z_2 are not equal, the chance that their last bits are equal "may be" approximately $\frac{1}{2}$. This heuristic argument is actually consistent with Theorem 1 when $r_1, r_2 \to 0$. According to (8), as $r_1, r_2 \to 0$, we have $A_{1,1}, A_{2,1} \to \frac{1}{2}$, and $C_{1,1}, C_{2,1} \to \frac{1}{2}$ also; and hence the probability (4) approaches $\frac{1+R}{2}$.

In practice, when a very accurate estimate is not necessary, one might actually use this approximate formula to simplify the estimator. The errors, however, could be quite noticeable when r_1, r_2 are not negligible; see Sec. 5.2.

2.2 The Unbiased Estimator

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

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

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

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. Following property of binomial distribution,

$$\text{Var}\left(\hat{R}_b\right) = \frac{\text{Var}\left(\hat{E}_b\right)}{\left[1-C_{2,b}\right]^2} = \frac{1}{k}\frac{E_b(1-E_b)}{\left[1-C_{2,b}\right]^2}$$
$$=\frac{1}{k}\frac{\left[C_{1,b}+(1-C_{2,b})R\right]\left[1-C_{1,b}-(1-C_{2,b})R\right]}{\left[1-C_{2,b}\right]^2} \qquad (11)$$

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

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

In fact, when $b \geq 32$, $\text{Var}\left(\hat{R}_b\right)$ and $\text{Var}\left(\hat{R}_M\right)$ are numerically indistinguishable for practical purposes.

2.3 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 the ***storage factor*** $B(b; R, r_1, r_2)$:

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

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 Theorem.

THEOREM 2. *If* $r_1 = r_2$ *and* $b_1 > b_2$, *then*

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

is a monotonically increasing function of $R \in [0, 1]$.

If $R \rightarrow 1$ *(which implies* $r_1 \rightarrow r_2$*), then*

$$\frac{B(b_1; R, r_1, r_2)}{B(b_2; R, r_1, r_2)} \rightarrow \frac{b_1}{b_2}\frac{1-A_{1,b_2}}{1-A_{1,b_1}}. \qquad (14)$$

If $r_1 = r_2$, $b_2 = 1$, $b_1 \geq 32$ *(hence we treat* $A_{1,b} = 0$*), then*

$$\frac{B(b_1; R, r_1, r_2)}{B(1; R, r_1, r_2)} = b_1\frac{R}{R+1-r_1} \qquad (15)$$

Proof: *We omit the proof due to its simplicity.*□

Suppose the original minwise hashing used 64 bits to store each sample, then the maximum improvement of b-bit minwise hashing would be 64-fold, attained when $r_1 = r_2 = 1$ and $R = 1$, according to (15). In the least favorable situation, i.e., $r_1, r_2 \rightarrow 0$, the improvement will still be $\frac{64}{3} = 21.3$-fold when $R = 0.5$.

Fig. 1 plots $\frac{B(64)}{B(b)}$, to directly visualize the relative improvements, which are consistent with what Theorem 2 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 Fig. 1.

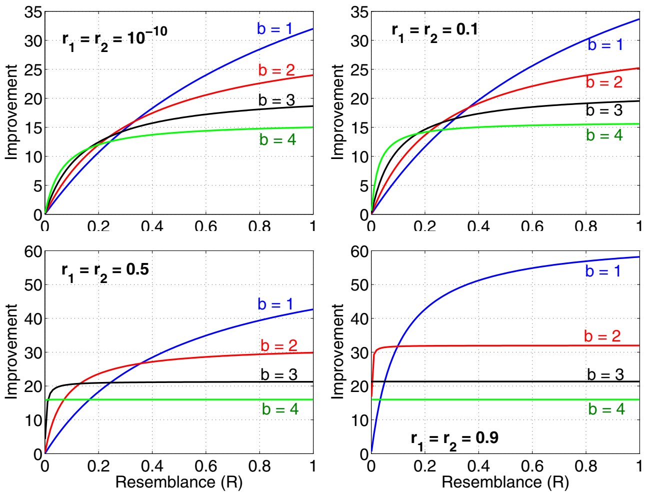

Figure 1: $\frac{B(64)}{B(b)}$, the relative storage improvement of using $b = 1, 2, 3, 4$ bits, compared to using 64 bits. $B(b)$ is defined in (12).

3. EXPERIMENTS

Experiment 1 is a sanity check, to verify: (A) our proposed estimator $\hat{R}_b$ in (9), is indeed unbiased; and (B) 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.

Experiment 3 is another duplicate detection task using 300,000 UCI NYTimes news articles.

3.1 Experiment 1

The data, extracted from Microsoft Web crawls, consists of 10 pairs of sets (i.e., total 20 words). Each set consists of the document IDs which contain the word at least once. Thus, this experiment is for estimating word associations.

Table 1: Ten pairs of words used in Experiment 1. For example, "KONG" and "HONG" correspond to the two sets of document IDs which contained word "KONG" and word "HONG" respectively.

Word 1	Word 2	r_1	r_2	R	$\frac{B(32)}{B(1)}$	$\frac{B(64)}{B(1)}$
KONG	HONG	0.0145	0.0143	0.925	15.5	31.0
RIGHTS	RESERVED	0.187	0.172	0.877	16.6	32.2
OF	AND	0.570	0.554	0.771	20.4	40.8
GAMBIA	KIRIBATI	0.0031	0.0028	0.712	13.3	26.6
UNITED	STATES	0.062	0.061	0.591	12.4	24.8
SAN	FRANCISCO	0.049	0.025	0.476	10.7	21.4
CREDIT	CARD	0.046	0.041	0.285	7.3	14.6
TIME	JOB	0.189	0.05	0.128	4.3	8.6
LOW	PAY	0.045	0.043	0.112	3.4	6.8
A	TEST	0.596	0.035	0.052	3.1	6.2

Table 1 summarizes the data and also provides the theoretical improvements, $\frac{B(32)}{B(1)}$ and $\frac{B(64)}{B(1)}$. The words were selected to include highly frequent word pairs (e.g., "OF-AND"), highly rare word pairs (e.g., "GAMBIA-KIRIBATI"), highly unbalanced pairs (e.g., "A-Test"), highly similar pairs (e.g, "KONG-HONG"), as well as word 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 $\hat{R}_b$ $(b = 1, 2, 3)$.

3.1.1 Validating the Unbiasedness

Figure 2 presents the estimation biases for the selected 2 word pairs. Theoretically, both estimators, $\hat{R}_M$ and $\hat{R}_b$, are unbiased (i.e., the y-axis in Figure 2 should be zero, after an infinite number of repetitions). Figure 2 verifies this fact because the empirical biases are all very small and no systematic biases can be observed.

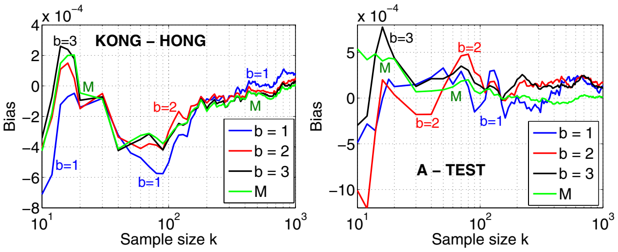

Figure 2: Empirical biases from 25000 simulations at each sample size k. "M" denotes the original minwise hashing.

3.1.2 Validating the Variance Formula

Figure 3 plots the empirical mean square errors (MSE = variance + bias^2) in solid lines, and the theoretical variances (11) in dashed lines, for 6 word pairs (instead of 10 pairs, due to the space limit).

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, MSE will differ from the variance).

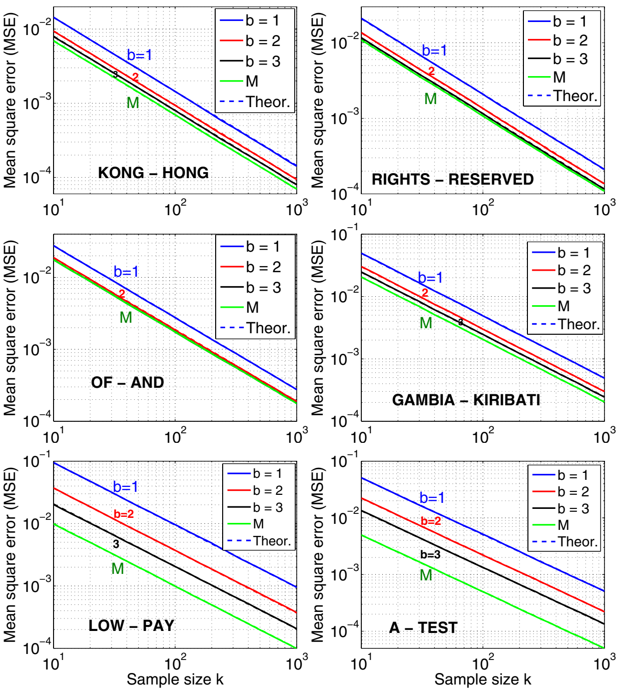

Figure 3: Mean square errors (MSEs). "M" denotes the original minwise hashing. "Theor." denotes the theoretical variances of Var$(\hat{R}_b)$(11) and Var$(\hat{R}_M)$(3). The dashed curves, however, are invisible because the empirical MSEs overlapped the theoretical variances. At the same k, Var$(\hat{R}_1)$ $>$ Var$(\hat{R}_2)$ $>$ Var$(\hat{R}_3)$ $>$ Var$(\hat{R}_M)$. However, $\hat{R}_1$ $(\hat{R}_2)$ only requires 1 bit (2 bits) per sample, while $\hat{R}_M$ requires 32 or 64 bits.

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 [15]. 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 (using 32 bits). 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.

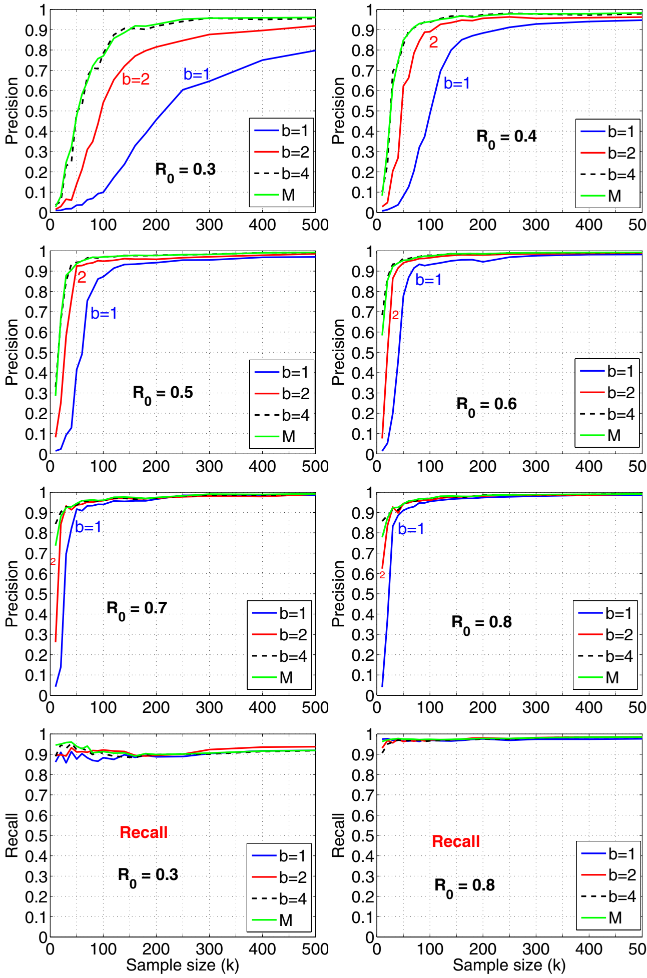

Figure 4: Microsoft collection of news data. The task is to retrieve news article pairs with resemblance $R \geq R_0$. The recall curves (bottom two panels) indicate all estimators are equally good (in recalls). The precision curves are more interesting for differentiating estimators. For example, when $R_0 = 0.4$ (top right panel), in order to achieve a precision = 0.80, the estimators $\hat{R}_M$, $\hat{R}_4$, $\hat{R}_2$, and $\hat{R}_1$ require $k = 50, 50, 75, 145$ samples, respectively, indicating $\hat{R}_4$, $\hat{R}_2$, and $\hat{R}_1$ respectively improve $\hat{R}_M$ by 8-fold, 10.7-fold, and 11-fold.

The precision curves for $\hat{R}_4$ (using 4 bits per sample) and $\hat{R}_M$ (using 32 bits per sample) are almost indistinguishable, suggesting a 8-fold improvement in space using $b = 4$.

When using $b = 1$ or 2, the space improvements are normally around 10-fold to 20-fold, compared to $\hat{R}_M$, especially for achieving high precisions (e.g., ≥ 0.9). This experiment again confirms the significant improvement of the b-bit minwise hashing using $b = 1$ (or 2). Table 2 summarizes the relative improvements.

In this experiment, $\hat{R}_M$ only used 32 bits per sample. For even larger applications, however, 64 bits per sample may be necessary [12]; and the improvements of $\hat{R}_b$ will be even more significant.

Note that in the context of (Web) document duplicate detection, in addition to shingling, a number of specialized hash-signatures have been proposed, which leverage properties of natural-language text (such as the placement of stopwords [31]). However, our approach is not aimed at any specific type of data, but is a general, domain-independent technique. Also, to the extent that other approaches rely on minwise hashing for signature computation, these may be combined with our techniques.

Table 2: Relative improvement (in space) of $\hat{R}_b$ (using b bits per sample) over $\hat{R}_M$ (32 bits per sample). For precision = 0.9, 0.95, we find the required sample sizes (from Figure 4) for $\hat{R}_M$ and $\hat{R}_b$ and use them to estimate the required storage in bits. The values in the table are the ratios of the storage costs. The improvements are consistent with the theoretical predictions in Figure 1.

R_0	Precision = 0.9			Precision = 0.95		
	$b = 1$	2	4	$b = 1$	2	4
0.3	—	5.7	8.8	—	—	7.1
0.4	9.2	10.0	8.3	—	10.0	8.2
0.5	10.8	12.7	8.4	8.2	10.1	7.7
0.6	12.9	11.7	8.6	10.5	12.4	8.5
0.7	16.0	14.8	9.6	15.4	12.7	7.6
0.8	17.4	10.3	8.0	18.7	14.2	7.7
0.9	16.6	14.0	10.7	23.0	17.6	9.7

3.3 Experiment 3: UCI NYTimes Data

We conducted another duplicate detection experiment on a public (UCI) collection of 300,000 NYTimes articles. The purpose is to ensure that our experiment will be repeatable by those who can not access the proprietary data in Experiment 2.

Figure 5 presents the precision curves for representative threshold R_0's. The recall curves are not shown because they could not differentiate estimators, just like in Experiment 1. The curves confirm again that using $b = 1$ or $b = 2$ bits, $\hat{R}_b$ could improve the original minwise hashing (using 32 bits per sample) by a factor of 10 or more. The curves for $\hat{R}_b$ with $b = 4$ almost always overlap with the curves for $\hat{R}_M$, verifying an expected 8-fold improvement.

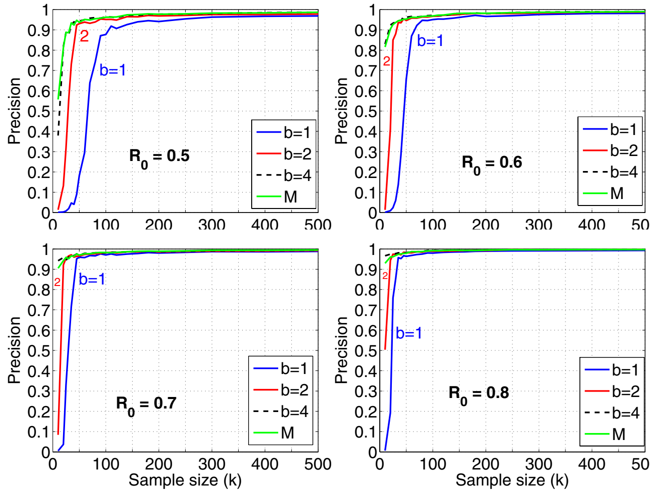

Figure 5: UCI collection of NYTimes data. The task is to retrieve news article pairs with resemblance $R \geq R_0$.

4. COMPARISONS WITH THE HAMMING DISTANCE LSH ALGORITHM

The *Hamming distance LSH* algorithm proposed in [20] is an influential 1-bit LSH scheme. In this algorithm, a set S_i, is mapped into a D-dimensional binary vector, y_i:

$$y_{it} = 1, \text{ if } t \in S_i; \quad y_{it} = 0, \text{ otherwise.}$$

k coordinates are randomly sampled from $\Omega = \{0, 1, ..., D-1\}$. We denote the samples of y_i by h_i, where $h_i = \{h_{ij}, j = 1 \text{ to } k\}$ is a k-dimensional vector. These samples will be used to estimate the Hamming distance H (using S_1, S_2 as an example):

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

Using the samples h_1 and h_2, an unbiased estimator of H is simply

$$\hat{H} = \frac{D}{k} \sum_{j=1}^{k} [h_{1j} \neq h_{2j}], \tag{16}$$

whose variance would be

$$\begin{aligned} \text{Var}\left(\hat{H}\right) =& \frac{D^2}{k^2} k \left[E\left([h_{1j} \neq h_{2j}]^2 \right) - E^2\left([h_{1j} \neq h_{2j}] \right) \right] \\ =& \frac{D^2}{k} \left[\frac{\sum_{i=0}^{D-1} [y_{1i} \neq y_{2i}]^2}{D} - \left(\frac{\sum_{i=0}^{D-1} [y_{1i} \neq y_{2i}]}{D} \right)^2 \right] \\ =& \frac{D^2}{k} \left[\frac{H}{D} - \frac{H^2}{D^2} \right]. \end{aligned} \tag{17}$$

The above analysis assumes $k \ll D$ (which is satisfied in practice); otherwise one should multiply the $\text{Var}\left(\hat{H}\right)$ in (17) by $\frac{D-k}{D-1}$, the "finite sample correction factor." It would be interesting to compare $\hat{H}$ with b-bit minwise hashing. In order to estimate H, we need to convert the resemblance estimator $\hat{R}_b$ (9) to $\hat{H}_b$:

$$\hat{H}_b = f_1 + f_2 - 2\frac{\hat{R}_b}{1 + \hat{R}_b}(f_1 + f_2) = \frac{1 - \hat{R}_b}{1 + \hat{R}_b}(f_1 + f_2). \tag{18}$$

The variance of $\hat{H}_b$ can be computed from $\text{Var}\left(\hat{R}_b\right)$ (11) using the "delta method" in statistics (note that $\left[\frac{1-x}{1+x}\right]' = \frac{-2}{(1+x)^2}$):

$$\begin{aligned} \text{Var}\left(\hat{H}_b\right) =& \text{Var}\left(\hat{R}_b\right)(f_1 + f_2)^2 \left(\frac{-2}{(1+R)^2} \right)^2 + O\left(\frac{1}{k^2}\right) \\ =& \text{Var}\left(\hat{R}_b\right) \frac{4(r_1 + r_2)^2}{(1+R)^4} D^2 + O\left(\frac{1}{k^2}\right). \end{aligned} \tag{19}$$

Recall $r_i = f_i/D$. To verify the variances in (17) and (19), we conduct experiments using the same data as in Experiment 1. This time, we estimate H instead of R, using both $\hat{H}$ (16) and $\hat{H}_b$ (18).

Figure 6 reports the mean square errors, together with the theoretical variances (17) and (19). We can see that the theoretical variance formulas are accurate. When the data is not dense, the estimator $\hat{H}_b$ (18) given by b-bit minwise hashing is much more accurate than the estimator $\hat{H}$ (16). However, when the data is dense (e.g., "OF-AND"), $\hat{H}$ could still outperform $\hat{H}_b$.

We now compare the actual storage needed by $\hat{H}_b$ and $\hat{H}$. We define the following two ratios to make fair comparisons:

$$W_b = \frac{\text{Var}\left(\hat{H}\right) \times k}{\text{Var}\left(\hat{H}_b\right) \times bk}, \quad G_b = \frac{\text{Var}\left(\hat{H}\right) \times \frac{r_1 + r_2}{2} 64k}{\text{Var}\left(\hat{H}_b\right) \times bk}. \tag{20}$$

W_b and G_b are defined in the same spirit as the ratio of the *storage factors* introduced in Sec. 2.3. Recall each sample of b-bit minwise hashing requires b bits (i.e., bk bits per set). If we assume each sample in the *Hamming distance* LSH requires 1 bit, then W_b in (20) is a fair indicator and $W_b > 1$ means $\hat{H}_b$ outperforms $\hat{H}$.

However, as can be verified in Fig. 6 and Fig 7, when r_1 and r_2 are small (which is usually the case in practice), W_b tends to be very

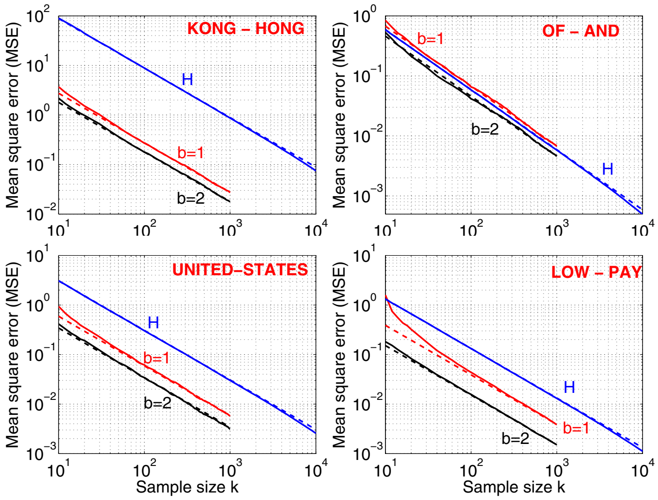

Figure 6: MSEs (normalized by H^2), for comparing $\hat{H}$ (16) with $\hat{H}_b$ (18). In each panel, three solid curves stand for $\hat{H}$ (labeled by "H"), $\hat{H}_1$ (by "b=1"), and $\hat{H}_2$ (by "b=2"), respectively. The dashed lines are the corresponding theoretical variances (17) and (19), which are largely overlapped by the solid lines. When the sample size k is not large, the empirical MSEs of $\hat{H}_b$ deviate from the theoretical variances, due to the bias caused by the nonlinear transformation of $\hat{H}_b$ from $\hat{R}_b$ in (18).

large, indicating a highly significant improvement of b-bit minwise hashing over the *Hamming distance LSH* algorithm in [20].

We consider in practice one will most likely implement the algorithm by only storing non-zero locations. In other words, for set S_i, only $r_i \times k$ locations need to be stored (each is assumed to use 64 bits). Thus, the total bits on average will be $\frac{r_1+r_2}{2}64k$ (per set). In fact, we have the following Theorem for G_b when $r_1, r_2 \to 0$.

THEOREM 3. *Consider $r_1, r_2 \to 0$, and G_b as defined in (20).*

$$\text{If } R \to 0, \quad \text{then } G_b \to \frac{8}{b}\left(2^b - 1\right). \tag{21}$$

$$\text{If } R \to 1, \quad \text{then } G_b \to \frac{64}{b}\frac{2^b-1}{2^b}. \tag{22}$$

Proof: *We omit the proof due to its simplicity.* □

Figure 7 plots W_1 and G_1, for $r_1 = r_2 = 10^{-6}, 10^{-4}$, 0.001, 0.01, 0,1 (which are probably reasonable in practice), as well as $r_1 = r_2 = 0.9$ (as a sanity check). Note that, not all combinations of r_1, r_2, R are possible. For example, when $r_1 = r_2 = 1$, then R has to be 1.

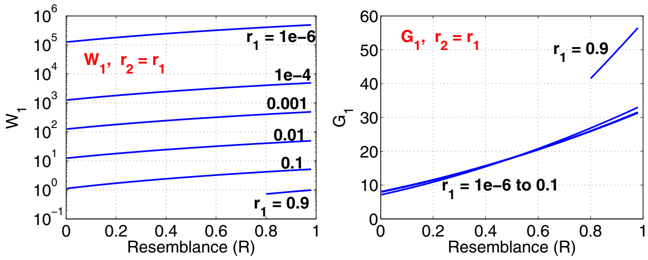

Figure 7: W_1 and G_1 as defined in (20). We consider $r_1 = 10^{-6}, 10^{-4}, 0.001, 0.01, 0.1, 0.9$. Note that not all combinations of (r_1, r_2, R) are possible. The plot for G_1 also verifies the theoretical limits proved in Theorem 3.

Figure 7 confirms our theoretical results. W_1 will be extremely large, when r_1, r_2 are small. However, when r_1 is very large (e.g., 0.9), it is possible that $W_1 < 1$, meaning that the *Hamming distance LSH* could still outperform b-bit minwise in dense data.

By only storing the non-zero locations, Figure 7 illustrates that b-bit minwise hashing will outperform the *Hamming distance LSH* algorithm, usually by a factor of 10 (for small R) to 30 (for large R and $r_1 \approx r_2$).

5. DISCUSSIONS

5.1 Computational Overhead

The previous results establish the significant reduction in storage requirements possible using b-bit minwise hashing. This section demonstrates that these also translate into significant improvements in computational overhead in the **estimation phrase**. The computational cost in the **preprocessing phrase**, however, will increase.

5.1.1 Preprocessing Phrase

In the preprocessing phrase, we need to generate minwise hashing functions and apply them to all the sets for creating fingerprints. This phrase is actually fairly fast [4] and is usually done off-line, incurring a one-time cost. Also, sets can be individually processed, meaning that this step is easy to parallelize.

The computation required for b-bit minwise hashes differs from the computation of traditional minwise hashes in two respects: (A) we require a larger number of (smaller-sized) samples, in turn requiring more hashing and (B) the packing of b-bit samples into 64-bit (or 32-bit) words requires additional bit-manipulation.

It turns out the overhead for (B) is small and the overall computation time scales nearly linearly with k; see Fig. 8. As we have analyzed, b-bit minwise hashing only requires increasing k by a small factor such as 3. Therefore, we consider the overhead in the preprocessing stage not to be a major issue. Also, it is important to note that b-bit minwise hashing provides the flexibility of trading storage with preprocessing time by using $b > 1$.

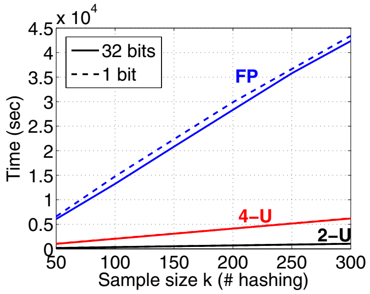

Figure 8: Running time in the preprocessing phrase on 100K news articles. 3 hashing functions were used: 2-universal hashing (labeled by "2-U"), 4-universal hashing (labeled by "4-U"), and full permutations (labeled by "FP"). Experiments with 1-bit hashing are reported in 3 dashed lines, which are only slightly higher (due to additional bit-packing) than their corresponding solid lines (the original minwise hashing using 32-bit).

The experiment in Fig. 8 was conducted on 100K articles from the BLEWS project [15]. We considered 3 hashing functions: first, 2-universal hash functions (computed using the fast universal hashing scheme described [10]); second, 4-universal hash-functions (computed using the CWtrick algorithm of [32]); and finally full random permutations (computed using the *Fisher-Yates shuffle* [13]).

5.1.2 Estimation Phrase

We have shown earlier that, when $R \geq 0.5$ and $b = 1$, we expect a storage reduction of at least a factor of 21.3, compared to using

64 bits. In the following, we will analyze how this impacts the computational overhead of the estimation.

Here, the key operation is the computation of the number of identical b-bit samples. While standard hash signatures that are multiples of 16-bit 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 stored in a compact manner, meaning that individual b-bit samples $e_{1,i,j}$ and $e_{2,i,j}$, $i = 1, \ldots, b$, $j = 1, \ldots k$ 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 an each offset h, we first compute $v = A_1[h] \oplus A_2[h]$, where $\oplus$ denotes the bitwise-XOR. Subsequently, the h-th bit of v will be set if and only if the h-th bits in $A_1[h]$ and $A_2[h]$ are different. Hence, to compute the number of overlapping b-bit samples encoded in $A_1[h]$ and $A_2[h]$, we need to compute the number of b-bit blocks ending at offsets divisible by b that only contain 0s.

The case of $b = 1$ corresponds to the problem of counting the number of 0-bits in a word. We tested different methods suggested in [34] 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. Then we can compute the number of 0-bits in v (in case of $w = 32$) as

$$c = bits[v \,\&\, \text{0xffffu}] + bits[(v \gg 16) \,\&\, \text{0xffffu}].$$

Interestingly, we can use the same method for the cases where $b > 1$, as we only need to modify the values stored in $bits$, setting $bits[i]$ to the number of b-bit blocks that only contain 0-bits in the binary representation of i.

We evaluated this approach using a loop computing the number of identical samples in two signatures 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.The results were essentially identical for $b = 2$.

Combined with the reduction in overall storage (for a given accuracy level), this means a significant speed improvement in the estimation phase: suppose in the original minwise hashing, each sample is stored using 64 bits. If we use 1-bit minwise hashing and consider $R > 0.5$, our previous analysis has shown that we could gain a storage reduction at least by a factor of 64/3 = 21.3 fold. The improvement in computational efficiency would be 21.3/1.67 = 12.8 fold, which is still significant.

5.2 Reducing Storage Overhead for r_1 and r_2

The unbiased estimator $\hat{R}_b$ (9) requires knowing $r_1 = \frac{f_1}{D}$ and $r_2 = \frac{f_1}{D}$. The storage cost could be a concern if r_1 (r_2) must be represented with a high accuracy (e.g., 64 bits).

This section illustrates that we only need to quantize r_1 and r_2 into Q levels, where $Q = 2^4$ is probably good enough and $Q = 2^8$ is more than sufficient. In other words, for each set, we only need to increase the total storage by 4 bits or 8 bits, which are negligible.

For simplicity, we carry out the analysis for $b = 1$ and $r_1 = r_2 = r$. In this case, $A_{1,1} = A_{2,1} = C_{1,1} = C_{2,1} = \frac{1-r}{2-r}$, and the correct estimator, denoted by $\hat{R}_{1,r}$ would be

$$\hat{R}_{1,r} = (2-r)\hat{E}_1 - (1-r),$$
$$Bias\left(\hat{R}_{1,r}\right) = E\left(\hat{R}_{1,r}\right) - R = 0,$$
$$\text{Var}\left(\hat{R}_{1,r}\right) = \frac{(1-r+R)(1-R)}{k}.$$

See the definition of $\hat{E}_1$ in (10). Now, suppose we only store an approximate value of r, denoted by $\tilde{r}$. The corresponding (approximate) estimator is denoted by $\hat{R}_{1,\tilde{r}}$:

$$\hat{R}_{1,\tilde{r}} = (2-\tilde{r})\hat{E}_1 - (1-\tilde{r}),$$
$$Bias\left(\hat{R}_{1,\tilde{r}}\right) = E\left(\hat{R}_{1,\tilde{r}}\right) - R = \frac{(\tilde{r}-r)(1-R)}{2-r},$$
$$\text{Var}\left(\hat{R}_{1,\tilde{r}}\right) = \frac{(1-r+R)(1-R)}{k}\frac{(2-\tilde{r})^2}{(2-r)^2}.$$

Thus, the (absolute) bias is upper bounded by $|\tilde{r}-r|$ (in the worst case, i.e., $R = 0$ and $r = 1$). Using $Q = 2^4$ levels of quantization, the bias is bounded by $1/16 = 0.0625$. In a reasonable situation, e.g., $R \geq 0.5$, the bias will be much smaller than 0.0625. Of course, if we increase the quantization levels to $Q = 2^8$, the bias ($< 1/256 = 0.0039$) will be negligible, even in the worst case.

Similarly, by examining the difference of the variances,

$$\left|\text{Var}\left(\hat{R}_{1,r}\right) - \text{Var}\left(\hat{R}_{1,\tilde{r}}\right)\right|$$
$$=\frac{|\tilde{r}-r|}{k}(1-r+R)(1-R)\frac{(4-\tilde{r}-r)}{(2-r)^2},$$

we can see that $Q = 2^8$ would be more than sufficient.

5.3 Combining Bits for Enhancing Performance

Our theoretical and empirical results have confirmed that, when the resemblance R is reasonably high, each bit per sample may contain strong information for 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

$$E_1 = \mathbf{Pr}\left(e_{1,1,\pi} = e_{2,1,\pi}\right) = C_{1,1} + \left(1 - C_{2,1}\right)R$$

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

$$T = \mathbf{Pr}\left(x_1 = x_2\right) = E_1^2 + \left(1 - E_1\right)^2, \tag{23}$$

which is a quadratic equation with a solution

$$R = \frac{\sqrt{2T-1} + 1 - 2C_{1,1}}{2 - 2C_{2,1}}. \tag{24}$$

We can estimate T without bias as a binomial. The resultant estimator for R will be biased, at small sample size k, due to the nonlinearity. We will recommend the following estimator

$$\hat{R}_{1/2} = \frac{\sqrt{\max\{2\hat{T}-1, 0\}} + 1 - 2C_{1,1}}{2 - 2C_{2,1}}. \tag{25}$$

The truncation $\max\{\ .\ , 0\}$ will introduce further bias; but it is necessary and is usually a good bias-variance trade-off. We use $\hat{R}_{1/2}$ to indicate that two bits are combined into one. The asymptotic variance of $\hat{R}_{1/2}$ can be derived using the "delta method"

$$\text{Var}\left(\hat{R}_{1/2}\right) = \frac{1}{k}\frac{T(1-T)}{4(1-C_{2,1})^2(2T-1)} + O\left(\frac{1}{k^2}\right). \tag{26}$$

Note that each sample is still stored using 1 bit, despite that we use "$b = 1/2$" to denote this estimator.

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

$$\lim_{R\rightarrow 1} \frac{\text{Var}\left(\hat{R}_1\right)}{\text{Var}\left(\hat{R}_{1/2}\right)} = \lim_{R\rightarrow 1} \frac{2(1-2E_1)^2}{(1-E_1)^2+E_1^2} = 2. \qquad (27)$$

(Recall, if $R = 1$, then $r_1 = r_2$, $C_{1,1} = C_{2,1}$, and $E_1 = C_{1,1} + 1 - C_{2,1} = 1$.) 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

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

Figure 9 plots the empirical MSEs for four 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 the fairly similar pair, "OF-AND," $\hat{R}_{1/2}$ is still considerably better. For "UNITED-STATES," whose $R = 0.591$, $\hat{R}_{1/2}$ performs similarly to $\hat{R}_1$. For "LOW-PAY," whose $R = 0.112$ only, the theoretical variance of $\hat{R}_{1/2}$ is very large. However, owing to the truncation in (25) (i.e., the variance-bias trade-off), the empirical performance of $\hat{R}_{1/2}$ is not too bad.

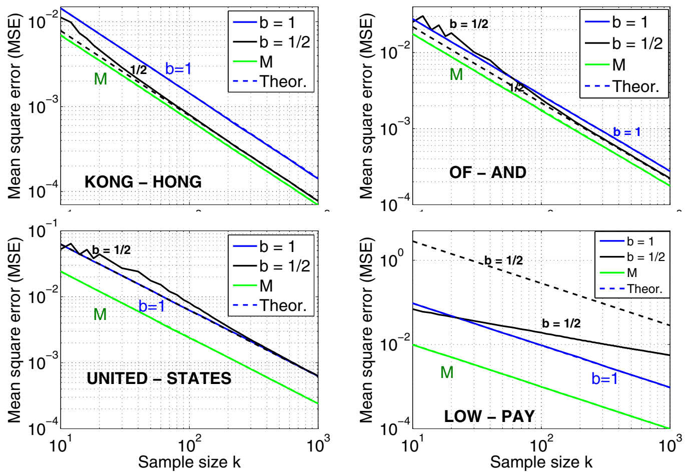

Figure 9: MSEs for comparing $\hat{R}_{1/2}$ (25) with $\hat{R}_1$ and $\hat{R}_M$. Due to the bias of $\hat{R}_{1/2}$, the theoretical variances Var $\left(\hat{R}_{1/2}\right)$, i.e., (26), deviate from the empirical MSEs when k is small.

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

6. CONCLUSION

The ***minwise hashing*** technique has been widely used as a standard duplicate detection approach in the context of information retrieval, for efficiently computing set similarity in massive data sets. Prior studies commonly used 64 bits to store each hashed value.

This study proposes ***b-bit minwise hashing***, by only storing the lowest b bits of each hashed value. We theoretically prove 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, compared to storing each hashed value using 64 bits. We also discussed the idea of combining 2 bits from different hashed values, to further enhance the improvement, when the target similarity is very high.

Our proposed method is simple and requires only minimal modification to the original minwise hashing algorithm. We expect our method will be adopted in practice.

7. 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, 2009.

[2] Michael Bendersky and W. Bruce Croft. Finding text reuse on the web. In *WSDM*, pages 262–271, 2009.

[3] Sergey Brin, James Davis, and Hector Garcia-Molina. Copy detection mechanisms for digital documents. In *SIGMOD*, pages 398–409, 1995.

[4] Andrei Z. Broder. On the resemblance and containment of documents. In *Sequences*, pages 21–29, 1997.

[5] 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.

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

[7] Gregory Buehrer and Kumar Chellapilla. A scalable pattern mining approach to web graph compression with communities. In *WSDM*, pages 95–106, 2008.

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

[9] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Michael Mitzenmacher, Alessandro Panconesi, and Prabhakar Raghavan. On compressing social networks. In *KDD*, pages 219–228, 2009.

[10] Dietzfelbinger, Martin and Hagerup, Torben and Katajainen, Jyrki and Penttonen, Martti A reliable randomized algorithm for the closest-pair problem. *Journal of Algorithms*, 25(1):19–51, 1997.

[11] Yon Dourisboure, Filippo Geraci, and Marco Pellegrini. Extraction and classification of dense implicit communities in the web graph. *ACM Trans. Web*, 3(2):1–36, 2009.

[12] D. Fetterly, M. Manasse, M. Najork, and J. Wiener. A large-scale study of the evolution of web pages. In *WWW*, pages 669–678, 2003.

[13] R.A. Fisher and F. Yates. *Statistical Tables for Biological, Agricultural and Medical Research.* Oliver & Boyd, 1948.

[14] 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.

[15] 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*, 2008.

[16] Aristides Gionis and Dimitrios Gunopulos and Nick Koudas. Efficient and Tunable Similar Set Retrieval. In *SIGMOD*, pages 247-258, 2001.

[17] Sreenivas Gollapudi and Aneesh Sharma. An axiomatic approach for result diversification. In *WWW*, pages 381–390, 2009.

[18] Monika .R. Henzinge. Algorithmic challenges in web search engines. *Internet Mathematics*, 1(1):115–123, 2004.

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

[20] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, 1998.

[21] 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, 2003.

[22] P. Li and K. Church. A Sketch Algorithm for Estimating Two-way and Multi-way Associations *Computational Linguistics*, pages 305–354, 2007. (Preliminary results appeared in HLT/EMNLP 2005.)

[23] P. Li, K. Church and T. Hastie. One Sketch For All: Theory and Applications of Conditional Random Sampling. In *NIPS*, 2008.

[24] Nitin Jindal and Bing Liu. Opinion spam and analysis. In *WSDM*, pages 219–230, 2008.

[25] Konstantinos Kalpakis and Shilang Tang. Collaborative data gathering in wireless sensor networks using measurement co-occurrence. *Computer Commu.*, 31(10):1979–1992, 2008.

[26] Eyal Kaplan, Moni Naor, and Omer Reingold. Derandomized constructions of k-wise (almost) independent permutations. *Algorithmica*, 55(1):113–133, 2009.

[27] Ludmila, Kave Eshghi, Charles B. Morrey III, Joseph Tucek, and Alistair Veitch. Probabilistic frequent itemset mining in uncertain databases. In *KDD*, pages 1087–1096, 2009.

[28] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting Near-Duplicates for Web-Crawling. In *WWW*, 2007.

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

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

[31] Martin Theobald, Jonathan Siddharth, and Andreas Paepcke. Spotsigs: robust and efficient near duplicate detection in large web collections. In *SIGIR*, pages 563–570, 2008.

[32] Mikkel Thorup and Yin Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In *SODA*, 2004.

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

[34] Henry S. Warren. *Hacker's Delight*. Addison-Wesley, 2002.

APPENDIX

A. PROOF OF THEOREM 1

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

$$z_1 = \min\left(\pi\left(S_1\right)\right), \qquad z_2 = \min\left(\pi\left(S_2\right)\right).$$

Define $e_{1,i} = i$th lowest bit of z_1, and $e_{2,i} = i$th lowest bit of z_2. The task is to derive $\mathbf{Pr}\left(\prod_{i=1}^{b} 1\{e_{1,i} = e_{2,i}\} = 1\right)$,

which can be decomposed to be

$$\begin{aligned}
&\mathbf{Pr}\left(\prod_{i=1}^{b} 1\{e_{1,i} = e_{2,i}\} = 1,\ z_1 = z_2\right)\\
&+\mathbf{Pr}\left(\prod_{i=1}^{b} 1\{e_{1,i} = e_{2,i}\} = 1,\ z_1 \neq z_2\right)\\
&=\mathbf{Pr}\left(z_1 = z_2\right) + \mathbf{Pr}\left(\prod_{i=1}^{b} 1\{e_{1,i} = e_{2,i}\} = 1,\ z_1 \neq z_2\right)\\
&=R + \mathbf{Pr}\left(\prod_{i=1}^{b} 1\{e_{1,i} = e_{2,i}\} = 1,\ z_1 \neq z_2\right).
\end{aligned}$$

where $R = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = \mathbf{Pr}\left(z_1 = z_2\right)$ is the resemblance.

When $b = 1$, the task boils down to estimating

$$\begin{aligned}
&\mathbf{Pr}\left(e_{1,1} = e_{2,1},\ z_1 \neq z_2\right)\\
&= \sum_{i=0,2,4,...}\left\{\sum_{j\neq i, j=0,2,4,...} \mathbf{Pr}\left(z_1 = i, z_2 = j\right)\right\}\\
&+ \sum_{i=1,3,5,...}\left\{\sum_{j\neq i, j=1,3,5,...} \mathbf{Pr}\left(z_1 = i, z_2 = j\right)\right\}.
\end{aligned}$$

Therefore, we need the following basic probability formula:

$$\mathbf{Pr}\left(z_1 = i,\ z_2 = j,\ i \neq j\right).$$

We start with

$$\mathbf{Pr}\left(z_1 = i,\ z_2 = j,\ i < j\right) = \frac{P_1 + P_2}{P_3}, \quad \text{where}$$

$$\begin{aligned}
P_3 &= \binom{D}{a}\binom{D-a}{f_1-a}\binom{D-f_1}{f_2-a},\\
P_1 &= \binom{D-j-1}{a}\binom{D-j-1-a}{f_2-a-1}\binom{D-i-1-f_2}{f_1-a-1},\\
P_2 &= \binom{D-j-1}{a-1}\binom{D-j-a}{f_2-a}\binom{D-i-1-f_2}{f_1-a-1}.
\end{aligned}$$

The expressions for P_1, P_2, and P_3 can be understood by the experiment of randomly throwing f_1+f_2-a balls into D locations, labeled $0, 1, 2, ..., D-1$. Those $f_1 + f_2 - a$ balls belong to three disjoint sets: $S_1 - S_1 \cap S_2$, $S_2 - S_1 \cap S_2$, and $S_1 \cap S_2$. Without any restriction, the total number of combinations should be P_3.

To understand P_1 and P_2, we need to consider two cases:

1. *The* j*th element is not in* $S_1 \cap S_2$*:* $\Longrightarrow P_1$.
   We first allocate the $a = |S_1 \cap S_2|$ overlapping elements randomly in $[j+1,\ D-1]$, resulting in $\binom{D-j-1}{a}$ combinations. Then we allocate the remaining $f_2 - a - 1$ elements in S_2 also randomly in the unoccupied locations in $[j+1, D-1]$, resulting in $\binom{D-j-1-a}{f_2-a-1}$ combinations. Finally, we allocate the remaining elements in S_1 randomly in the unoccupied locations in $[i+1, D-1]$, which has $\binom{D-i-1-f_2}{f_1-a-1}$ combinations.
2. *The* j*th element is in* $S_1 \cap S_2$*:* $\Longrightarrow P_2$.

After conducing expansions and cancelations, we obtain

$$\begin{aligned}
&\mathbf{Pr}\left(z_1 = i,\ z_2 = j,\ i < j\right) = \frac{P_1 + P_2}{P_3}\\
&= \frac{\left(\frac{1}{a} + \frac{1}{f_2-a}\right)\frac{(D-j-1)!(D-i-1-f_2)!}{(a-1)!(f_1-a-1)!(f_2-a-1)!(D-j-f_2)!(D-i-f_1-f_2+a)!}}{\frac{D!}{a!(f_1-a)!(f_2-a)!(D-f_1-f_2+a)!}}\\
&= \frac{f_2(f_1-a)(D-j-1)!(D-f_2-i-1)!(D-f_1-f_2+a)!}{D!(D-f_2-j)!(D-f_1-f_2+a-i)!}\\
&= \frac{f_2(f_1-a)\prod_{t=0}^{j-i-2}(D-f_2-i-1-t)\prod_{t=0}^{i-1}(D-f_1-f_2+a-t)}{\prod_{t=0}^{j}(D-t)}\\
&= \frac{f_2}{D}\frac{f_1-a}{D-1}\prod_{t=0}^{j-i-2}\frac{D-f_2-i-1-t}{D-2-t}\prod_{t=0}^{i-1}\frac{D-f_1-f_2+a-t}{D+i-j-1-t}
\end{aligned}$$

For convenience, we introduce the following notation:

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

Also, we assume D is large (which is always satisfied in practice). Thus, we can obtain a reasonable approximation:

$$\begin{aligned}
&\mathbf{Pr}\left(z_1 = i,\ z_2 = j,\ i < j\right)\\
&= r_2(r_1 - s)\left[1 - r_2\right]^{j-i-1}\left[1 - (r_1 + r_2 - s)\right]^{i}
\end{aligned}$$

Similarly, we obtain, for large D,

$$\begin{aligned}
&\mathbf{Pr}\left(z_1 = i,\ z_2 = j,\ i > j\right)\\
&= r_1(r_2 - s)\left[1 - r_1\right]^{i-j-1}\left[1 - (r_1 + r_2 - s)\right]^{j}
\end{aligned}$$

Now we have the tool to calculate the probability

$$\begin{aligned}
&\mathbf{Pr}\left(e_{1,1} = e_{2,1},\ z_1 \neq z_2\right)\\
&= \sum_{i=0,2,4,...}\left\{\sum_{j\neq i, j=0,2,4,...} \mathbf{Pr}\left(z_1 = i, z_2 = j\right)\right\}\\
&+ \sum_{i=1,3,5,...}\left\{\sum_{j\neq i, j=1,3,5,...} \mathbf{Pr}\left(z_1 = i, z_2 = j\right)\right\}
\end{aligned}$$

For example, (again, assuming D is large)

$$\begin{aligned}
&\mathbf{Pr}\left(z_1 = 0, z_2 = 2, 4, 6, ...\right)\\
&= r_2(r_1 - s)\left([1 - r_2] + [1 - r_2]^3 + [1 - r_2]^5 + ...\right)\\
&= r_2(r_1 - s)\frac{1 - r_2}{1 - [1 - r_2]^2}
\end{aligned}$$

$$\begin{aligned}\mathbf{Pr}\left(z_1=1, z_2=3,5,7,...\right) = &r_2(r_1-s)[1-(r_1+r_2-s)] \\ &\times \left([1-r_2]+[1-r_2]^3+[1-r_2]^5+...\right) \\ =&r_2(r_1-s)[1-(r_1+r_2-s)]\frac{1-r_2}{1-[1-r_2]^2}.\end{aligned}$$

Therefore,

$$\begin{aligned}&\sum_{i=0,2,4,...}\left\{\sum_{i<j,j=0,2,4,...}\mathbf{Pr}\left(z_1=i,z_2=j\right)\right\} \\ &+\sum_{i=1,3,5,...}\left\{\sum_{i<j,j=1,3,5,...}\mathbf{Pr}\left(z_1=i,z_2=j\right)\right\} \\ =&r_2(r_1-s)\frac{1-r_2}{1-[1-r_2]^2}\times \\ &\left(1+[1-(r_1+r_2-s)]+[1-(r_1+r_2-s)]^2+...\right) \\ =&r_2(r_1-s)\frac{1-r_2}{1-[1-r_2]^2}\frac{1}{r_1+r_2-s}.\end{aligned}$$

By symmetry, we know

$$\begin{aligned}&\sum_{j=0,2,4,...}\left\{\sum_{i>j,i=0,2,4,...}\mathbf{Pr}\left(z_1=i,z_2=j\right)\right\} \\ &+\sum_{j=1,3,5,...}\left\{\sum_{i>j,i=1,3,5,...}\mathbf{Pr}\left(z_1=i,z_2=j\right)\right\} \\ =&r_1(r_2-s)\frac{1-r_1}{1-[1-r_1]^2}\frac{1}{r_1+r_2-s}.\end{aligned}$$

Combining the probabilities, we obtain

$$\begin{aligned}&\mathbf{Pr}\left(e_{1,1}=e_{2,1},\ z_1\neq z_2\right) \\ =&\frac{r_2(1-r_2)}{1-[1-r_2]^2}\frac{r_1-s}{r_1+r_2-s}+\frac{r_1(1-r_1)}{1-[1-r_1]^2}\frac{r_2-s}{r_1+r_2-s} \\ =&A_{1,1}\frac{r_2-s}{r_1+r_2-s}+A_{2,1}\frac{r_1-s}{r_1+r_2-s},\end{aligned}$$

where

$$A_{1,b}=\frac{r_1\left[1-r_1\right]^{2^b-1}}{1-\left[1-r_1\right]^{2^b}},\qquad A_{2,b}=\frac{r_2\left[1-r_2\right]^{2^b-1}}{1-\left[1-r_2\right]^{2^b}}.$$

Therefore, we can obtain the desired probability, for $b=1$,

$$\begin{aligned}&\mathbf{Pr}\left(\prod_{i=1}^{b=1}1\{e_{1,i}=e_{2,i}\}=1\right) \\ =&R+A_{1,1}\frac{r_2-s}{r_1+r_2-s}+A_{2,1}\frac{r_1-s}{r_1+r_2-s} \\ =&R+A_{1,1}\frac{f_2-a}{f_1+f_2-a}+A_{2,1}\frac{f_1-a}{f_1+f_2-a} \\ =&R+A_{1,1}\frac{f_2-\frac{R}{1+R}(f_1+f_2)}{f_1+f_2-\frac{R}{1+R}(f_1+f_2)}+A_{2,1}\frac{f_1-a}{f_1+f_2-a} \\ =&R+A_{1,1}\frac{f_2-Rf_1}{f_1+f_2}+A_{2,1}\frac{f_1-Rf_2}{f_1+f_2} \\ =&C_{1,1}+(1-C_{2,1})R\end{aligned}$$

where

$$\begin{aligned}C_{1,b}&=A_{1,b}\frac{r_2}{r_1+r_2}+A_{2,b}\frac{r_1}{r_1+r_2} \\ C_{2,b}&=A_{1,b}\frac{r_1}{r_1+r_2}+A_{2,b}\frac{r_2}{r_1+r_2}.\end{aligned}$$

To this end, we have proved the main result for $b=1$.

Next, we consider $b>1$. Due to the space limit, we only provide a sketch of the proof. When $b=2$, we need

$$\begin{aligned}&\mathbf{Pr}\left(e_{1,1}=e_{2,1},e_{1,2}=e_{2,2},\ \ z_1\neq z_2\right) \\ =&\sum_{i=0,4,8,...}\left\{\sum_{j\neq i,j=0,4,8,...}\mathbf{Pr}\left(z_1=i,z_2=j\right)\right\} \\ &+\sum_{i=1,5,9,...}\left\{\sum_{j\neq i,j=1,5,9,...}\mathbf{Pr}\left(z_1=i,z_2=j\right)\right\} \\ &+\sum_{i=2,6,10,...}\left\{\sum_{j\neq i,j=2,6,10,...}\mathbf{Pr}\left(z_1=i,z_2=j\right)\right\} \\ &+\sum_{i=3,7,11,...}\left\{\sum_{j\neq i,j=3,7,11,...}\mathbf{Pr}\left(z_1=i,z_2=j\right)\right\}\end{aligned}$$

We again use the basic probability formula $\mathbf{Pr}\left(z_1=i,z_2=j,i<j\right)$ and the sum of (different) geometric series, for example,

$$[1-r_2]^3+[1-r_2]^7+[1-r_2]^{11}+...=\frac{[1-r_2]^{2^2-1}}{1-[1-r_2]^{2^2}}.$$

Similarly, for general b, we will need

$$[1-r_2]^{2^b-1}+[1-r_2]^{2\times 2^b-1}+[1-r_2]^{3\times 2^b-1}+...=\frac{[1-r_2]^{2^b-1}}{1-[1-r_2]^{2^b}}.$$

After more algebra, we prove the general case:

$$\begin{aligned}&\mathbf{Pr}\left(\prod_{i=1}^{b}1\{e_{1,i}=e_{2,i}\}=1\right) \\ =&R+A_{1,b}\frac{r_2-s}{r_1+r_2-s}+A_{2,b}\frac{r_1-s}{r_1+r_2-s} \\ =&C_{1,b}+(1-C_{2,b})R,\end{aligned}$$

It remains to show some useful properties of $A_{1,b}$ (same for $A_{2,b}$). The first derivative of $A_{1,b}$ with respect to b is

$$\begin{aligned}\frac{\partial A_{1,b}}{\partial b}=&\frac{r_1[1-r_1]^{2^b-1}\log(1-r_1)\log 2\left(1-[1-r_1]^{2^b}\right)}{\left(1-[1-r_1]^{2^b}\right)^2} \\ &-\frac{-[1-r_1]^{2^b}\log(1-r_1)\log 2\, r_1\left(1-[1-r_1]^{2^b-1}\right)}{\left(1-[1-r_1]^{2^b}\right)^2} \\ \leq&0\qquad\qquad\text{(Note that }\log(1-r_1)\leq 0)\end{aligned}$$

Thus, $A_{1,b}$ is a monotonically decreasing function of b. Also,

$$\lim_{r_1\rightarrow 0}A_{1,b}=\lim_{r_1\rightarrow 0}\frac{[1-r_1]^{2^b-1}-r_1\left(2^b-1\right)[1-r_1]^{2^b-2}}{2^b[1-r_1]^{2^b-1}}=\frac{1}{2^b},$$

$$\begin{aligned}\frac{\partial A_{1,b}}{\partial r_1}=&\frac{[1-r_1]^{2^b-1}-r_1\left(2^b-1\right)[1-r_1]^{2^b-2}}{\left(1-[1-r_1]^{2^b}\right)} \\ &-\frac{2^b[1-r_1]^{2^b-1}r_1\left[1-r_1\right]^{2^b-1}}{\left(1-[1-r_1]^{2^b}\right)^2} \\ =&\frac{[1-r_1]^{2^b-2}}{\left(1-[1-r_1]^{2^b}\right)^2}\left(1-2^b r_1-[1-r_1]^{2^b}\right)\leq 0.\end{aligned}$$

Note that $(1-x)^c\geq 1-cx$, for $c\geq 1$ and $x\leq 1$. Therefore $A_{1,b}$ is a monotonically decreasing function of r_1.