Set similarity with b-bit k-permutation Minwise Hashing - DRAFT

Simon Wehrli

10.5.2013

1 Introduction

This report is written for the seminar titled "Algorithms for Database Systems" at ETH Zürich. The seminar participants read and summarize various papers, which treat solving problems in the context of Big Data, which is this year's topic. This report summarizes and explains the core concepts of [1] and [2].

With Big Data, most problems emphasis shifts from single computational complexity to memory space usage considerations. Typically, the runtime is dominated by the time needed for memory accesses, hence the goal is to reduce the number of memory accesses. Often this is achieved by more compact data structures and at the price of less accurate result.

Many today's applications are faced with very large datasets. A common task is to find *similarity* between two or several such sets. There are lots of problem solutions which use a mapping to sets of properties and then do a *similarity search* on them. Fast (approximative) algorithms for this search enable improvements of many well-known algorithms, e.g. of machine learning and computer vision.

We start by explaining the original MINWISE HASHING, which nicely presents the basic idea behind all algorithms. We move on to a major improvement, the B-BIT K-PERMUTATION MINWISE HASHING, which reduces storage at the cost of more iterations in the algorithm and is a generalization of the concept. We then present ONE PERMUTATION HASHING, which achieves surprising accuracy using only one permutation. We focus on the concepts and algorithms and will reference to papers for applications in practice.

1.1 Similarity

We denote by Ω the set of all possible items of the sets $S_n \subseteq \Omega$, $n = 1$ to N. $|\Omega| = D$ is always large (e.g. $D = 2^{64}$). Often we consider only two sets $S_1 = X$, $S_2 = Y$. Let $a = |X \cap Y|$, $b = |X| - a$ and $c = |Y| - a$. Figure 1 on the following page summarizes these definitions graphically.

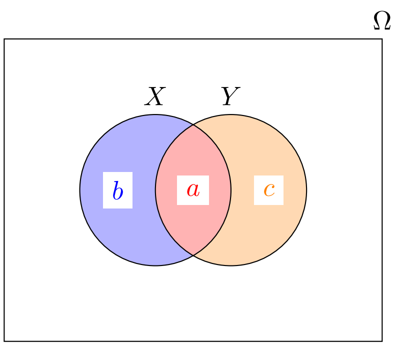

Figure 1: Two example sets in Ω and the notation a, b, c for the sizes of important subsets.

Definition 1. *The normalized similarity between two sets X and Y, known as* resemblance *or* Jaccard similarity, *denoted by R, is*

$$R = \frac{|X \cap Y|}{|X \cup Y|} = \frac{a}{|X| + |Y| - a} = \frac{a}{a+b+c} \tag{1}$$

Other Notations Later we often look at some important event of a problem where we use $\Pr[\cdot]$ as a shorthand for the *Probability* of that event. To get an estimator for R out of a probabilistic argument, we always use some Bernoulli experiment. This is a process, where we repeatedly flip a possibly biased coin, but the bias does not change. We can see it as a sequence of binary random variables, because only two outcomes are possible, 0 or 1. An event is described by an equation. We use the notation $1\{equation\}$ which is one if and only if the *equation* in curly braces is true:

$$1\{equation\} = \begin{cases} 1 & \text{if } equation \text{ evaluates to } true \\ 0 & \text{otherwise} \end{cases}$$

1.2 Motivating Example

Consider a web search provider, which want to present a result list of web pages without duplicates. To achieve that, for every pair of web pages, we drop one of them, if they are textually very similar. This is the case when their *resemblance* R is greater than some threshold R_0. But to be able to use the *resemblance* as a measurement, we have to map each page to a set. One could imagine to define this mapping from the page to the set of all words occurring on that page. But with this mapping would not keep track of the order in which their appear. As in several studies [3, 4] we will instead map a page to a set of *shingles*. A shingle is a string of w contiguous words, and we include a *shingle* in our result set if the *shingle* occurs on the page (in the same order). Typically we choose $w = 5$. Figure 2 on the next page shows an exemplary mapping.

$$\boxed{\begin{array}{l}\text{This web}\\ \text{page is just}\\ \text{an example.}\end{array}} \xrightarrow{\text{mapping}} \begin{array}{l}\{\text{“This web page”},\\ \text{“web page is”},\\ \text{“page is just”},\\ \text{“is just an”},\\ \text{“just an example”}\}\end{array}$$

Figure 2: The mapping from a web page to a set of *shingles*.

Clearly, the number of possible shingles and therefore D is huge. Assuming 10^5 different English words, we have $D = \left(10^5\right)^5 \gg 2^{64}$. Thus computing the exact similarities for all pairs of pages of a web search would require prohibitive storage. In general we would already need $\Theta(D)$ storage for a single pair. We need approximation algorithms to improve significantly on this bound.

2 Original Minwise Hashing

A working approximative solution to this problem was described by Broder and his colleagues [3, 4]. We demonstrate their algorithm on a small running example. Suppose we have two web pages P_X and P_Y and want to compare them:

$$\boxed{\text{u v u v}} \xleftrightarrow{\text{similar?}} \boxed{\text{w v u}}$$

We map both pages to sets of length-2-shingles. So P_X becomes {“uv”, “vu”} and P_Y becomes {“wv”, “vu”}. Note that the mapping is indifferent to that “uv” occurs twice on P_X. When we continuously number all possible shingles {“uu”, “uv”, “uw”, “vu”, “vv”, “vw”, “wu”, “wv”, “ww”} and define this to be Ω_{shingle}, we can visualize sets in a binary matrix

$$M \in \{0,1\}^{N\times D}, \qquad M_{ni} = 1\{i \in S_n\}.$$

Applied to our example, we get

$$M = \begin{array}{r} \text{Sets:} \\ X: \\ Y: \end{array} \overset{\Omega_{\text{shingle}}}{\overbrace{\begin{array}{ccccccccc} \text{uu} & \text{uv} & \text{uw} & \text{vu} & \text{vv} & \text{vw} & \text{wu} & \text{wv} & \text{ww} \end{array}}} \atop \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}. \tag{2}$$

Because working with an Ω_{shingle} of strings is nasty, we assume there is a perfect hash function applied to the elements of the original domain which always gives us $\Omega = \{0, 1, \ldots, D-1\}$. Additionally we colour the columns of the matrix according to the three coloured subsets in Figure 1 on the preceding page, e.g.

a column with both entries equals to one represents an element contained in both sets. M is rewritten with the new Ω as

$$M = \begin{array}{r|ccccccccc} & \multicolumn{9}{c}{\overbrace{\hspace{12em}}^{\Omega}} \\ \text{Sets:} & 0&1&2&3&4&5&6&7&8 \\ X: & 0&1&0&1&0&0&0&0&0 \\ Y: & 0&0&0&1&0&0&0&1&0 \\ & & b & & a & & & & c & \end{array}. \tag{3}$$

For reasons which later become clear, suppose a random permutation π is performed on Ω[1],

$$\pi : \Omega \longrightarrow \Omega.$$

To simplify notation, we overload the definition of π to work also for subsets of Ω. Thus with $\pi(S_n)$ we denote the application of the permutation π to every element of the set S_n. More precisely,

$$\begin{aligned} &\pi : 2^{\Omega} \longrightarrow 2^{\Omega}, \\ &\pi : S_n \longmapsto \pi\left(S_n\right) = \left\{\pi\left(i\right) | i \in S_n\right\}. \end{aligned}$$

In the matrix representation, the application of π can be seen as rearranging the matrix columns in a random order. For an exemplary chosen π this could look like

$$M' = \begin{array}{r|ccccccccc} & \multicolumn{9}{c}{\overbrace{\hspace{12em}}^{\pi\left(\Omega\right)}} \\ \text{Sets:} & 0&1&2&3&4&5&6&7&8 \\ \pi(X): & 0&0&0&0&0&1&0&0&1 \\ \pi(Y): & 0&0&1&0&0&1&0&0&0 \\ & & & c & & & a & & & b \\ & \triangle & \triangle & \blacktriangle & & & & & & \end{array}. \tag{4}$$

Suppose now there is a pointer $\blacktriangle$ starting at the left most column of M', moving to the right until it sees the first column where there is at least one 1 (i.e. skip columns of only zeros). In our example $\blacktriangle$ would stop at index 2. We can ask for the probability of the event ω, that $\blacktriangle$ stops at a column with two ones, which corresponds to the subset a:

$$\Pr\left[\blacktriangle \text{ stops at } \begin{pmatrix} 1 \\ 1 \end{pmatrix}\right] = \Pr\left[\omega\right] = \frac{a}{a+b+c} = R,$$

because we randomly permuted the columns. To determine if or if not ω arises for the given permutation π, we only need the index of the left most 1 in each row respectively the smallest element of each set S_n permuted with π. We define this as

$$h_{S_n,\pi} = min(\pi(S_n)).^{2} \tag{5}$$

[1]Note that in the paper [1], the hash function is applied after the permutation and the minimum-function, but it is simpler to understand this way.

[2]Note that the smallest element changes under the permutations because π is a permutation on Ω and S_n is only a subset of Ω in general.

This key observation leads to the crucial

Lemma 1. *For any two sets $X, Y \subseteq \Omega$*

$$\Pr[\min(\pi(X)) = \min(\pi(Y))] = \Pr[h_{X,\pi} = h_{Y,\pi}] = \frac{|X \cap Y|}{|X \cup Y|} = R. \tag{6}$$

To estimate R we must evaluate ω not only on one but k independent random permutations $\pi_1, \pi_2, \ldots, \pi_k$. By averaging the outcomes we finally build the unbiased estimator

$$\hat{R}_M = \frac{1}{k} \sum_{j=1}^{k} 1 \left\{ h_{X,\pi_j} = h_{Y,\pi_j} \right\}, \tag{7}$$

$$Var(\hat{R}_M) = \frac{1}{k} R(1 - R). \tag{8}$$

We later refer to h_{S_n,π_j} as a **sample** and to k as the **sample size**. Note that we only have to store a **sample** for each permutation and this for each set. But we can reuse the same permutations for all sets and precompute the k **samples** for each set individually. The minimum function $\min(\cdot)$ needs $O(D)$ time. For a similarity calculation of two specific sets we then only have to make the computations from (7), thus $O(k)$ steps. For the precomputation especially the storage requirement for the samples is relevant. Altogether we learn that the algorithm has to consist of two phases, the precomputation and the similarity calculation.

Many applications, especially duplicate detection, are interested in detecting somewhat high similarity, thus an approximation seems reasonable. From (8) we learn, that the accuracy can be adjusted by choosing the **sample size** appropriately. In practice we typically have a k between 2^5 to 2^8.

However, finding duplicates out of m objects, e.g. web pages, still needs $O(m^2)$ comparisons. There are number of approaches to deal with that problem, but we will not discuss it here and refer to [4, 5].

2.1 The Algorithm

Based on the theoretical results, Algorithm 1 on the next page presents the procedure of (k-permutation) Minwise Hashing.

Algorithm 1 Original Minwise Hashing algorithm, applied to estimating pairwise resemblances in a collection of N sets.

```
Input: Sets $S_n \subseteq \Omega = \{0, 1, \ldots, D-1\}, n = 1$ to $N$.          $\triangleright D = |\Omega|$
Output: Estimated resemblance $\hat{R}_M$
  // Pre-processing
  Generate $k$ random permutations $\pi_j : \Omega \longrightarrow \Omega, j = 1$ to $k$
  for all $n = 1$ to $N, j = 1$ to $k$ do
      Store $\min(\pi_j(S_n))$, denoted by $h_{S_n,\pi_j}$.
  end for

  // Estimation (Use two sets $X, Y$ as an example)
  Estimate the resemblance by $\hat{R}_M = \frac{1}{k}\sum_{j=1}^{k} 1\left\{h_{X,\pi_j} = h_{Y,\pi_j}\right\}$
```

3 b-bit k-permutation Minwise Hashing

We will present an algorithm which improves on the storage requirements of the original Minwise Hashing. The idea is to reduce the size of each **sample** by only taking b bits of it, as opposed to, e.g. 64 bits. Intuitively, this will increase the estimation variance $Var(\hat{R}_M)$, at the same **sample size** k. To maintain the same accuracy, we have to increase k. One can show, if the resemblance is not too small, we will not have to increase k much and in total use less storage.

For example, when $b = 1$ and $R = 0.5$, the estimation variance will increase at most by a factor of 3. To keep the same accuracy, we have to increase the **sample size** by a factor of 3. If we before stored each minimum $\min(\pi_j(S_n))$ using 64 bits, the improvement with $b = 1$ is $64/3 = 21.3$.

Consider again two sets $X, Y \subseteq \Omega$, on which a random permutation $\pi : \Omega \longrightarrow \Omega$ is applied. We extend our notation with

$$h_{X,b,\pi} = b \text{ lowest bits of } \min(\pi(X))$$

$$h_{Y,b,\pi} = b \text{ lowest bits of } \min(\pi(Y))$$

Theorem 1. *Assume D is large.*

$$P_b = \Pr[1\{h_{X,b,\pi} = h_{Y,b,\pi}\}] = C_{1,b} + (1 - C_{2,b})R \tag{9}$$

$$r_X = \frac{|X|}{D}, \qquad r_Y = \frac{|Y|}{D} \tag{10}$$

$$\begin{aligned} C_{1,b} &= A_{X,b}\frac{r_Y}{r_X + r_Y} + A_{Y,b}\frac{r_X}{r_X + r_Y}, \\ C_{2,b} &= A_{X,b}\frac{r_X}{r_X + r_Y} + A_{Y,b}\frac{r_Y}{r_X + r_Y} \end{aligned} \tag{11}$$

$$A_{X,b} = \frac{r_X[1 - r_X]^{2^b - 1}}{1 - [1 - r_X]^{2^b}}, \qquad A_{Y,b} = \frac{r_Y[1 - r_Y]^{2^b - 1}}{1 - [1 - r_Y]^{2^b}} \tag{12}$$

The intuition for the additional terms $C_{1,b}$ and $C_{2,b}$ in (9) on the following page compared to (6) on page 5 is that we have to account for a type of "false positive": When two minima agree on their last b bits, $h_{X,b,\pi} = h_{Y,b,\pi}$, it's still possible that their are different, $h_{X,\pi} \neq h_{Y,\pi}$. Thus even when $R = 0$, the collision probability P_b is not zero, but rather $C_{1,b}$. This makes the derivation much more complicated.[3]

Even though D is assumed to be large, experiments show that even for $D = 20$ the absolute error caused by using (9) on the previous page ist < 0.01.

3.1 The Estimator

From (9) on the preceding page of Theorem 1 on the previous page we derive the estimator $\hat{R}_b$ for R:

$$\hat{R}_b = \frac{\hat{P}_b - C_{1,b}}{1 - C_{2,b}} \tag{13}$$

$$\hat{P}_b = \frac{1}{k}\sum_{j=1}^{k} 1\left\{h_{X,b,\pi_j} = h_{Y,b,\pi_j}\right\} \tag{14}$$

This estimator is unbiased, i.e. $\mathrm{E}[\hat{R}_b] = R$. Furthermore, the variance of $\hat{R}_M$ converges to the variance of $\hat{R}_b$, i.e.

$$\lim_{b\to\inf} \mathrm{Var}\left(\hat{R}_b\right) = \frac{R(1-R)}{k} = \mathrm{Var}\left(\hat{R}_M\right) \tag{15}$$

3.2 The Algorithm

Based on the theoretical results, Algorithm 2 presents the procedure of b-bit (k-permutation) Minwise Hashing.

Algorithm 2 b-bit Minwise Hashing algorithm, applied to estimating pairwise resemblances in a collection of N sets.

```
Input: Sets $S_n \subseteq \Omega = \{0, 1, \ldots, D-1\}, n = 1$ to $N$.        ▷ $D = |\Omega|$
Output: Estimated resemblance $\hat{R}_b$
  // Pre-processing
  Generate $k$ random permutations $\pi_j : \Omega \longrightarrow \Omega, j = 1$ to $k$
  for all $n = 1$ to $N, j = 1$ to $k$ do
      Store the lowest $b$ bits of $\min(\pi_j(S_n))$, denoted by $h_{S_n,b,\pi_j}$.
  end for

  // Estimation (Use two sets $X, Y$ as an example)
  Compute $\hat{P}_b = \frac{1}{k}\sum_{j=1}^{k} 1\{h_{X,b,\pi_j} = h_{Y,b,\pi_j}\}$
  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 on the previous page
```

[3] A proof of Theorem 1 on the preceding page can be found in the appendix of [6].

3.3 Deriving the Hamming Distance

Another well-known measurement for the similarity is the *hamming distance*. For the purpose of calculating the *hamming distance* between two sets $X, Y \subseteq \Omega = \{0, 1, \ldots, D-1\}$, the sets are first mapped to a D-dimensional binary vector x, y resp.:

Definition 2. *Let vector* $x, y \in \{0,1\}^D$, $x_i = 1\{i \in X\}, y_i = 1\{i \in Y\}$. *The* hamming distance *between* X *and* Y *is*

$$H = \sum_{i=0}^{D-1} [x_i \neq y_i] = |X \cup Y| - |X \cap Y| = |X| + |Y| - 2a \tag{16}$$

If we reformulate 1 on page 2 as

$$a = \frac{R}{1+R}(|X| + |Y|), \tag{17}$$

we can use the b-bit Minwise Hashing algorithm to estimate H with

$$\hat{H}_b = |X| + |Y| - 2\frac{\hat{R}_b}{1+\hat{R}_b}(|X| + |Y|) = \frac{1-\hat{R}_b}{1+\hat{R}_b}(|X| + |Y|) \tag{18}$$

Experiments show that this approach is significantly faster than standard methods for computing the *hamming distance*.

3.4 Drawbacks

The major problem of b-bit Minwise Hashing is the costly preprocessing. Consider an application in machine learning, where the sets S_n represents some properties of an object (e.g. a document or an image). Often one wants to add objects dynamically at runtime and compare them to other objects by finding the *resemblance* of their properties. Finding the k minima under the permutations may take too long, i.e. uses $O(k|S_n|)$ to $O(kD)$ time per set. In general, the costly preprocessing may cause problems in user-facing applications, where the testing efficiency for new data objects is crucial. There is the need for a entirely fast algorithm to keep this applications responsive.

Another drawback is that storing k permutations is sometimes impractical. If e.g. $D = 10^9$, one permutation vector uses 4GB, which is still possible to store. But the space needed to store e.g. $k = 500$ permutation vectors (each of length D), is not tolerable.

4 One Permutation Hashing

This algorithm is directly motivated by the optimization potential of the standard Minwise Hashing method: intuitively, it ought to be "wasteful" in that all elements in a set are permuted, scanned but only the minimum will be used. As the name already suggests, we reduce the preprocessing step to only one permutation.

As in section 3.3 we will represent sets $S_n \subseteq \Omega$ as vectors $s_n \in \{0,1\}^D$, $(s_n)_i = 1\{i \in S_n\}$. We will setup a running example with $X, Y, Z \subseteq \Omega = \{0, 1, \cdots, 15\}$. Let be π some random permutation on Ω and the already permuted sets be

$$\pi(X) = \{2, 4, 7, 13\}, \quad \pi(Y) = \{0, 3, 6, 13\}, \quad \pi(Z) = \{0, 1, 10, 12\}.$$

Now again we build up a data matrix where the rows are equal to the vector representations of the permuted sets:

$$\begin{array}{r|cccccccccccccccc|}
 & \multicolumn{16}{c}{\overbrace{\hspace{22em}}^{\Omega}} \\
\text{Sets:} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\pi(X): & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\pi(Y): & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\pi(Z): & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0
\end{array} \tag{19}$$

The idea is to divide the columns evenly into t (here $t = 4$) bins (parts), take the minimum in each bin. Because later we only compare minima within one bin, we can re-index the elements to use the smallest possible representation:

$$\begin{array}{r|cccc|cccc|cccc|cccc|}
 & \multicolumn{4}{c}{\overbrace{\hspace{5em}}^{\Omega'}} & \multicolumn{4}{c}{\overbrace{\hspace{5em}}^{\Omega'}} & \multicolumn{4}{c}{\overbrace{\hspace{5em}}^{\Omega'}} & \multicolumn{4}{c}{\overbrace{\hspace{5em}}^{\Omega'}} \\
\text{Sets:} & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\
\pi(X): & 0 & 0 & \textcircled{1} & 0 & \textcircled{1} & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \textcircled{1} & 0 & 0 \\
\pi(Y): & \textcircled{1} & 0 & 0 & 1 & 0 & 0 & \textcircled{1} & 0 & 0 & 0 & 0 & 0 & 0 & \textcircled{1} & 0 & 0 \\
\pi(Z): & \textcircled{1} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \textcircled{1} & 0 & \textcircled{1} & 0 & 0 & 0 \\
 & \multicolumn{4}{l}{\text{bin 1}} & \multicolumn{4}{l}{\text{bin 2}} & \multicolumn{4}{l}{\text{bin 3}} & \multicolumn{4}{l}{\text{bin 4}}
\end{array} \tag{20}$$

We get the minima-vectors

$$\begin{aligned} v_X &= [2, 0, *, 1], \\ v_Y &= [0, 2, *, 1], \\ v_Z &= [0, *, 2, 0], \end{aligned} \tag{21}$$

where '$*$' denotes an empty bin.

To derive the *resemblance* between two sets, e.g. X, Y, we introduce two definitions:

$$\begin{aligned} \text{number of “jointly empty bins”:} \qquad N_{emp} &= \sum_{j=1}^{t} I_{emp,j}, \\ \text{number of “matched bins”:} \qquad N_{mat} &= \sum_{j=1}^{t} I_{mat,j}, \end{aligned} \tag{22}$$

where $I_{emp,j}$ and $I_{mat,j}$ are defined for the j-th bin, as

$$
\begin{aligned}
I_{emp,j} &= \begin{cases} 1 & \text{if both } \pi(X) \text{ and } \pi(Y) \text{ are empty in the } j\text{-th bin} \\ 0 & \text{otherwise} \end{cases} \\
I_{mat,j} &= \begin{cases} 1 & \text{if both } \pi(X) \text{ and } \pi(Y) \text{ are not empty and the smallest} \\ & \quad \text{elements in the } j\text{-th bin matches, i.e. } (v_X)_j = (v_Y)_j \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
\tag{23}
$$

4.1 The Estimator

Recall the notations $a = |X \cap Y|$ and $f = |X \cup Y| = |X| + |Y| - a$. We formulate the estimator as

Lemma 2. *The* resemblance *is estimated by*

$$\hat{R}_{mat} = \frac{N_{mat}}{k - N_{emp}} \tag{24}$$

is unbiased, i.e.

$$\mathrm{E}\left[\hat{R}_{mat}\right] = R. \tag{25}$$

In our example we have $N_{emp} = 1$ and $N_{mat} = 1$. Thus $\hat{R}_{mat} = 1/3$.

Because it is a bit surprising that the estimator is unbiased, we give a

Proof of Lemma 2 on the following page. Because we assume that the sets and thus the data vectors are not completely empty, it holds

$$t - N_{emp} > 0 \Rightarrow P\left[t - N_{emp} > 0\right] = 1,$$

hence we get rid of division-by-zero problems and $m > 0$ in (28) is always true. From the definitions in (23) follows

$$I_{emp,j} = 1 \Rightarrow I_{mat,j} = 0, \tag{26}$$

$$\mathrm{E}\left[I_{mat,j} | I_{emp,j} = 0\right] = R, \tag{27}$$

$$\text{for } m > 0: \quad \mathrm{E}\left[I_{mat,j} | t - N_{emp} = m\right] = \frac{m}{t} R, \tag{28}$$

and we derive

$$
\begin{aligned}
\mathrm{E}\left[N_{mat} | t - N_{emp} = m\right] &\overset{(26)}{=} \sum_{j=1}^{t-N_{emp}} \mathrm{E}\left[I_{mat,j} | t - N_{emp} = m\right] \\
&\overset{(27)}{=} \sum_{j=1}^{t-N_{emp}} R \qquad = R\left(t - N_{emp}\right).
\end{aligned}
\tag{29}
$$

Finally,

$$\begin{aligned} \mathrm{E}\left[\frac{N_{mat}}{t-N_{emp}}\middle| t-N_{emp}=m\right] &\overset{(29)}{=} R, \qquad \text{(independent of } N_{emp}) \\ \Rightarrow \mathrm{E}\left[\frac{N_{mat}}{t-N_{emp}}\right] &= \mathrm{E}\left[\hat{R}_{mat}\right] = R \end{aligned} \tag{30}$$

□

We give the variance without a proof.[4]

$$\mathrm{Var}\left[\hat{R}_{mat}\right] = R(1-R)\left(\mathrm{E}\left[\frac{1}{t-N_{emp}}\right]\left(1+\frac{1}{f-1}\right)-\frac{1}{f-1}\right) \tag{31}$$

If we have very few empty bins, i.e. N_{emp} is essentially zero and $t \ll f$, the term simplifies to

$$\mathrm{Var}\left[\hat{R}_{mat}\right] \approx \frac{R(1-R)}{t}\left(\frac{f-t}{f-1}\right) \approx \frac{R(1-R)}{t} \tag{32}$$

as expected.

4.2 The Algorithm

Based on the theoretical results, Algorithm 3 on the next page presents the procedure of One Permutation Hashing.

Algorithm 3 One Permutation Hashing algorithm, applied to estimating pairwise resemblances in a collection of N sets.

```
Input: Sets $S_n \subseteq \Omega = \{0,1,\dots,D-1\}, n = 1$ to $N$.        ▷ $D = |\Omega|$
Output: Estimated resemblance $\hat{R}_{mat}$
  // Pre-processing
  Generate one random permutations $\pi : \Omega \longrightarrow \Omega$
  for all $n = 1$ to $N$ do
      Permute set $S_n$ with $\pi$, store the re-indexed minima of each of the $t$ bins
  in a data vector $v_{S_n} \in \{0,\cdots,\lfloor |S_n|/t \rfloor\}^t$.        ▷ see (20)
  end for

  // Estimation (Use two sets $X, Y$ as an example)
  Estimate the resemblance by $\hat{R}_{mat} = \frac{N_{mat}}{t-N_{emp}}$        ▷ see (22)
```

Empirical results show that the One Permutation Hashing scheme performs as well or even slightly better than the b-bit k-permutation hashing scheme.

Let's have a look at the runtime and storage requirements and compare them with Algorithm 2 on page 7. The pre-processing step generates one permutation vector, which uses $O(D\log(D))$ storage, which even for large D is still

[4]A proof can be found in the appendix of [2].

practical (in contrast to storing k permutation vectors for the Minwise Hashing method). The pre-processing also generates a data vector with t elements (which uses themselves only $O(\log(D/t))$ bits because we re-indexed them within the bins) for each set, thus $O(t \log(D/t))$ bits per set.

Table 1 on the next page summarizes the runtime and storage requirements of the different algorithms.

Space used for ...	storing permutation(s)	storing pre-processing data per set
Naïve approach (store whole sets)	—	$O(D)$
original Minwise Hashing	$O(kD \log(D))$	$O(k \log(D))$
b-bit Minwise Hashing	$O(kD \log(D))$	$O(kb)$
One Permutation Hashing	$O(D \log(D))$	$O(t \log(D/t))$

Table 1: Algorithm comparison. We only list the space complexity because the processing time boundaries are given by the time needed to read the stored data at least once and hence is dominated by the space complexity.

5 Conclusions

Minwise hashing and One Permutation Hashing are standard techniques for efficiently estimating set similarity in massive datasets. We started at explaining the original Minwise Hashing and demonstrated how reducing the amount of stored bits per **sample** to b bits leads to a effective reduction of the required storage and consequently computational overhead.

We then went over to One Permutation Hashing, which uses the same basic idea, but makes better use of the permuted set and therefore only needs one permutation. Overall, this last algorithm is superior to the others in many aspects.

References

[1] Ping Li 0001 and Arnd Christian König. Theory and applications of b -bit minwise hashing. *Commun. ACM*, 54(8):101–109, 2011.

[2] Ping Li 0001, Art B. Owen, and Cun-Hui Zhang. One permutation hashing for efficient search and learning. *CoRR*, abs/1208.1259, 2012.

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

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

[5] Moses Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC-02)*, pages 380–388, New York, May 19–21 2002. ACM Press.

[6] Ping Li 0001 and Arnd Christian König. b-bit minwise hashing. *CoRR*, abs/0910.3349, 2009.