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

Simon Wehrli

28.4.2013

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 computational complexity to memory space usage considerations. Typically, the runtime is dominated by the time needed for memory accesses, hence optimizing the memory footprint already improves our algorithms significantly.

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 $S_i \subseteq \Omega$, $i = 1$ to n. $|\Omega| = D$ is always large (e.g. $D = 2^{64}$). Often we consider only two sets $S_1 = X$, $S_2 = Y$.

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

$$R = \frac{|X \cap Y|}{|X \cup Y|} = \frac{a}{|X| + |Y| - a}, \qquad \text{where } a = |X \cap Y| \tag{1}$$

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 the page. 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 1 shows an exemplary mapping.

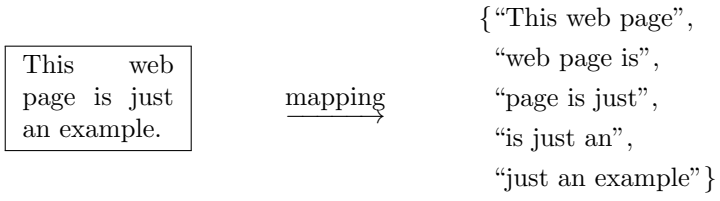

Figure 1: The mapping from a web page to a set of *shingles*.

Clearly, the number of possible shingles and therefore D is huge. Assume 10^5 different English words, we have $D = \left(10^5\right)^5 \in \Theta\left(2^{25}\right)$. Thus computing the exact similarities for all pairs of pages of a web search would require prohibitive storage. For a single pair, we would already need $\Theta(D)$ storage.

1.3 Original Minwise Hashing

A working approximative solution to this problem was described by Broder and his colleagues [3, 4]. Their algorithm is based on the following observation:

Suppose a random permutation π is performed on Ω[1],

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

With $\pi(S_i)$ we denote the application of the permutation π to every element of S_i. A key role play the minima of the permuted sets:

$$e_{S_i,\pi} = min(\pi(S_i))$$

We use $\Pr[\cdot]$ as a shorthand for the *Probability* of some event. An elementary probability argument shows that for two sets $X, Y \subseteq \Omega$

$$\Pr[e_{X,\pi} = e_{Y,\pi}] = \Pr[\min(\pi(X)) = \min(\pi(Y))] = \frac{|X \cap Y|}{|X \cup Y|} = R. \tag{2}$$

We can now build an unbiased estimator $\hat{R}_M$ of R with k minwise independent permutations, $\pi_1, \pi_2, \ldots, \pi_k$:

$$\hat{R}_M = \frac{1}{k} \sum_{j=1}^{k} 1 \left\{ e_{X,\pi_j} = e_{Y,\pi_j} \right\}, \tag{3}$$

$$Var(\hat{R}_M) = \frac{1}{k} R(1-R). \tag{4}$$

We later refer to e_{S_i,π_j} as a **sample** and to k as the **sample size**. To store a **sample** may require e.g., 64 bits. Note that the minimum function $\min(\cdot)$ needs $O(D)$ time and constant space and can be precomputed for each set S_i and permutation π_j.[2]

Many applications, especially duplicate detection, are interested in detecting somewhat high similarity, thus an approximation seems reasonable. From equation (4) we learn, that the accuracy can be adjusted by choosing the **sample size** appropriately.

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 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

[1] We assume there is a perfect hash function applied to the elements of the original domain which always gives us $\Omega = \{0, 1, \ldots, D-1\}$. 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] A detailed runtime analysis will follow for the improved algorithm in section 2.4 on page 6

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 $\min(\pi_j(S_i))$ 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

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

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

Theorem 1. *Assume D is large.*

$$P_b = \Pr[1\{e_{X,b,\pi} = e_{Y,b,\pi}\}] = C_{1,b} + (1 - C_{2,b})R \tag{5}$$

$$r_X = \frac{|X|}{D}, \qquad r_Y = \frac{|Y|}{D} \tag{6}$$

$$\begin{aligned} C_{1,b} &= A_{X,b}\frac{r_Y}{r_X + r_Y} + A_{Y,b}\frac{r_X}{r_X + r_Y}, \\ C_{2,b} &= A_{X,b}\frac{r_X}{r_X + r_Y} + A_{Y,b}\frac{r_Y}{r_X + r_Y} \end{aligned} \tag{7}$$

$$A_{X,b} = \frac{r_X[1 - r_X]^{2^b - 1}}{1 - [1 - r_X]^{2^b}}, \qquad A_{Y,b} = \frac{r_Y[1 - r_Y]^{2^b - 1}}{1 - [1 - r_Y]^{2^b}} \tag{8}$$

The intuition for the additional terms $C_{1,b}$ and $C_{2,b}$ in (5) compared to (2) on the preceding page is that we have to account for a type of "false positive": When two minima agree on their last b bits, $e_{X,b,\pi} = e_{Y,b,\pi}$, it's still possible that their are different, $e_{X,\pi} \neq e_{Y,\pi}$. 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 (5) ist < 0.01.

2.1 The Estimator

From (5) of Theorem 1 we derive the estimator $\hat{R}_b$ for R:

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

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

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

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.

$$\lim_{b\to\inf} \mathrm{Var}\left(\hat{R}_b\right) = \frac{R(1-R)}{k} = \mathrm{Var}\left(\hat{R}_M\right) \tag{11}$$

2.2 The Algorithm

Based on the theoretical results, Algorithm 1 presents the procedure of b-bit (k-permutation) Hashing.

Algorithm 1 b-bit Minwise Hashing algorithm, applied to estimating pairwise resemblances in a collection of n sets.

Input: Sets $S_i \subseteq \Omega = \{0, 1, \ldots, D-1\}, i = 1$ to n. ▷ $D = |\Omega|$
Output: Estimated resemblance $\hat{R}_b$

```
// Pre-processing
Generate k random permutations π_j : Ω ⟶ Ω, j = 1 to k
for all i = 1 to n, j = 1 to k do
    Store the lowest b bits of min(π_j(S_i)), denoted by e_{S_i,b,π_j}.
end for

// Estimation (Use two sets X, Y as an example)
Compute P̂_b = (1/k) Σ_{j=1}^{k} 1{e_{X,b,π_j} = e_{Y,b,π_j}}
Estimate the resemblance by R̂_b = (P̂_b − C_{1,b}) / (1 − C_{2,b}), where C_{1,b} and C_{2,b} are from
Theorem 1 on the preceding page
```

2.3 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* $x, y \in \{0,1\}^D$, $x_t = 1\{t \in X\}, y_t = 1\{t \in Y\}$. *The* hamming distance *between* X *and* Y *is*

$$H = \sum_{i=0}^{D-1} [x_i \neq y_i] = |X \cup Y| - |X \cap Y| = |X| + |Y| - 2a \tag{12}$$

If we reformulate 1 on page 2 as

$$a = \frac{R}{1+R}(|X| + |Y|), \tag{13}$$

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

$$\hat{H}_b = |X| + |Y| - 2\frac{\hat{R}_b}{1+\hat{R}_b}(|X| + |Y|) = \frac{1-\hat{R}_b}{1+\hat{R}_b}(|X| + |Y|) \tag{14}$$

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

2.4 Drawbacks

The major problem of b-bit Minwise Hashing is the costly preprocessing. Consider an application in machine learning, where the sets S_i 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 $O(k|S_i|)$ 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. But the space needed to store e.g. $k = 500$ permutation vectors (each of length D), is not tolerable.

3 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 2.3 on the preceding page we will represent sets $S_i \subseteq \Omega$ as vectors $s_i \in \{0,1\}^D$, $(s_i)_t = 1\{t \in S_i\}$. We will setup a running example with $X, Y, Z \subseteq \Omega = \{0, 1, \cdots, 15\}$. Let

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

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

$$\begin{array}{l} \\ \text{Sets:} \\ \pi(X): \\ \pi(Y): \\ \pi(Z): \end{array} \overbrace{\begin{array}{cccccccccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ \hline 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{array}}^{\Omega} \tag{15}$$

The idea is to divide the columns evenly into k (here $k = 4$) 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:

$$
\begin{array}{l}
\text{Sets:} \\
\pi(X): \\
\pi(Y): \\
\pi(Z):
\end{array}
\overbrace{\begin{array}{cccc} 0 & 1 & 2 & 3 \end{array}}^{\Omega'}
\overbrace{\begin{array}{cccc} 0 & 1 & 2 & 3 \end{array}}^{\Omega'}
\overbrace{\begin{array}{cccc} 0 & 1 & 2 & 3 \end{array}}^{\Omega'}
\overbrace{\begin{array}{cccc} 0 & 1 & 2 & 3 \end{array}}^{\Omega'}
\left(\begin{array}{cccc|cccc|cccc|cccc}
0 & 0 & \textcircled{1} & 0 & \textcircled{1} & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \textcircled{1} & 0 & 0 \\
\textcircled{1} & 0 & 0 & 1 & 0 & 0 & \textcircled{1} & 0 & 0 & 0 & 0 & 0 & 0 & \textcircled{1} & 0 & 0 \\
\textcircled{1} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \textcircled{1} & 0 & \textcircled{1} & 0 & 0 & 0
\end{array}\right)
\begin{array}{cccc} \text{bin 1} & & \text{bin 2} & \end{array}
\begin{array}{cc} \text{bin 3} & \text{bin 4} \end{array}
\tag{16}
$$

We get the minima-vectors

$$
\begin{aligned}
v_X &= [2, 0, *, 1], \\
v_Y &= [0, 2, *, 1], \\
v_Z &= [0, *, 2, 0],
\end{aligned}
\tag{17}
$$

where '$*$' denotes an empty bin.

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

$$
\begin{aligned}
\text{number of ``jointly empty bins'':} \qquad & N_{emp} = \sum_{j=1}^{k} I_{emp,j}, \\
\text{number of ``matched bins'':} \qquad & N_{mat} = \sum_{j=1}^{k} I_{mat,j},
\end{aligned}
\tag{18}
$$

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

$$
\begin{aligned}
I_{emp,j} &= \begin{cases} 1 & \text{if both } \pi(X) \text{ and } \pi(Y) \text{ are empty in the } j\text{-th bin} \\ 0 & \text{otherwise} \end{cases} \\
I_{mat,j} &= \begin{cases} 1 & \text{if both } \pi(X) \text{ and } \pi(Y) \text{ are not empty and the smallest} \\ & \quad \text{elements in the } j\text{-th bin matches, i.e. } (v_X)_j = (v_Y)_j \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
\tag{19}
$$

3.1 The Estimator

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

Lemma 1. *The* resemblance *is estimated by*

$$
\hat{R}_{mat} = \frac{N_{mat}}{k - N_{emp}} \tag{20}
$$

is unbiased, i.e.

$$
\mathrm{E}\left[\hat{R}_{mat}\right] = R. \tag{21}
$$

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 1 on the previous page. Because we assume that the sets and thus the data vectors are not completely empty, it holds

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

hence we get rid of division-by-zero problems and $m > 0$ in (24) is always true. From the definitions in (19) on the previous page follows

$$I_{emp,j} = 1 \Rightarrow I_{mat,j} = 0, \tag{22}$$

$$\mathrm{E}\left[I_{mat,j} | I_{emp,j} = 0\right] = R, \tag{23}$$

$$\text{for } m > 0: \quad \mathrm{E}\left[I_{mat,j} | k - N_{emp} = m\right] = \frac{m}{k} R, \tag{24}$$

and we derive

$$\begin{aligned} \mathrm{E}\left[N_{mat} | k - N_{emp} = m\right] &\overset{(22)}{=} \sum_{j=1}^{k-N_{emp}} \mathrm{E}\left[I_{mat,j} | k - N_{emp} = m\right] \\ &\overset{(23)}{=} \sum_{j=1}^{k-N_{emp}} R \qquad = R\left(k - N_{emp}\right). \end{aligned} \tag{25}$$

Finally,

$$\begin{aligned} \mathrm{E}\left[\frac{N_{mat}}{k - N_{emp}} \middle| k - N_{emp} = m\right] &\overset{(25)}{=} R, \qquad \text{(independent of } N_{emp}) \\ \Rightarrow \mathrm{E}\left[\frac{N_{mat}}{k - N_{emp}}\right] &= \mathrm{E}\left[\hat{R}_{mat}\right] = R \end{aligned} \tag{26}$$

□

We give the variance without a proof.[4]

$$\mathrm{Var}\left[\hat{R}_{mat}\right] = R(1-R)\left(\mathrm{E}\left[\frac{1}{k - N_{emp}}\right]\left(1 + \frac{1}{f-1}\right) - \frac{1}{f-1}\right) \tag{27}$$

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

$$\mathrm{Var}\left[\hat{R}_{mat}\right] \approx \frac{R(1-R)}{k}\left(\frac{f-k}{f-1}\right) \approx \frac{R(1-R)}{k} \tag{28}$$

as expected.

3.2 The Algorithm

Based on the theoretical results, Algorithm 2 on the following page presents the procedure of One Permutation Hashing.

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

Algorithm 2 ONE PERMUTATION HASHING algorithm, applied to estimating pairwise resemblances in a collection of n sets.

Input: Sets $S_i \subseteq \Omega = \{0, 1, \ldots, D-1\}, i = 1$ to n. ▷ $D = |\Omega|$
Output: Estimated resemblance $\hat{R}_{mat}$

// Pre-processing
Generate one random permutations $\pi : \Omega \longrightarrow \Omega$
for all $i = 1$ to n **do**
Permute set S_i with π, store the re-indexed minima of each of the k bins in a data vector $v_{S_i} \in \{0, \cdots, \lfloor |S_i| / k \rfloor\}^k$. ▷ see (16)
end for

// Estimation (Use two sets X, Y as an example)
Estimate the resemblance by $\hat{R}_{mat} = \frac{N_{mat}}{k - N_{emp}}$ ▷ see (18)

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 1 on page 5. The pre-processing step generates one permutation vector, which uses $\Theta(D)$ storage, which even for large D is still practical (in contrast to storing k permutation vectors for the MINWISE HASHING method). The pre-processing also generates a data vector with k elements (which uses themselves only $O(\log(D/k))$ bits because we re-indexed them within the bins) for each set, thus $O(k \log(D/k))$ bits per set.

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

Space/Time, Step	B-BIT MINWISE HASHING	ONE PERMUTATION HASHING
Space for storing permutation(s)	$O(kD \log(D))$	$O(D \log(D))$
Time complexity pre-processing per set	$O(kD)$	$O(D)$
Space for storing pre-processing data per set	$O(kb)$	$O(k \log(D/k))$
Time complexity estimating similarity of two sets	$O(kb)$	$O(k \log(D/k))$

Table 1: Algorithm comparison. Note that k denotes different things for the tow algorithms, the number of permutations resp. the number of bins. Because typically $k \leq 512$, we could also consider it as a constant.

4 Conclusions

MINWISE HASHING and ONE PERMUTATION HASHING are standard techniques for efficiently estimating set similarity in massive datasets. We started at ex-

plaining 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.