PERMUTATION GROUPING: INTELLIGENT HASH FUNCTION DESIGN FOR AUDIO & IMAGE RETRIEVAL

Shumeet Baluja, Michele Covell and Sergey Ioffe
Google Research, Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043

ABSTRACT

The combination of MinHash-based signatures and Locality-Sensitive Hashing (LSH) schemes has been effectively used for finding approximate matches in very large audio and image retrieval systems. In this study, we introduce the idea of *permutation-grouping* to intelligently design the hash functions that are used to index the LSH tables. This helps to overcome the inefficiencies introduced by hashing real-world data that is noisy, structured, and most importantly is not independently and identically distributed. Through extensive tests, we find that permutation-grouping dramatically increases the efficiency of the overall retrieval system by lowering the number of low-probability candidates that must be examined by 30-50%.

Index Terms— Audio Retrieval, Image Retrieval, LSH, MinHash

1. INTRODUCTION

Hashing is one of the most common ways to perform efficient lookups in large databases, but suffers from the fact that a small perturbation of the data point can dramatically change the hash value. This makes hashing using a single hash function a poor candidate for nearest neighbor (NN) computation. Locality Sensitive Hashing (LSH) addresses the approximate-NN problem by using multiple hash functions [5]. Consider L groups of B randomly created hash functions. Given a data point, we compute L keys, each of which is the concatenation of B hash values. By hashing both the reference and the probe into L tables, we restrict the search to only the examples for which at least one of the keys – all B hash values – match. To find the approximate matches to a probe, we perform L lookups (one from each table), and take the union of the resulting candidate sets. In its simplest form, the candidate for which the largest number of hash groups (out of L) matched the probe is the best match. Assuming the hash-table lookups take constant time, LSH lookups take $O(L)$ time per probe. LSH has been effectively applied to the retrieval of approximate-duplicate matches in the audio, video and image domains [1][2][7].

Recently, a system was created based on a combination of using *MinHash Signatures* [4] to describe both audio and video data, and an LSH approach for retrieval [1]. It was designed to hold 10^8-10^9 keys, distributed to a network of machines. Despite this success and the theoretical guarantees that can be made about MinHash+LSH [4][5], severe inefficiencies were encountered with respect to the required computation and bandwidth. These inefficiencies arose because the expected performance was computed with respect to the $L*B$ hash functions, and ignored the distribution of the data itself. In reality, most data generates non-uniform, highly-correlated distributions. With correlated distributions, the probability of randomly generating "unlucky" hash functions increases dramatically. Individual hash groups, from our set of L groups, can be non-distinctive: They will map many dissimilar examples to the same hash-bin, leading to excessive numbers of candidates from each lookup. Long candidate lists increase computation cost (to tally the evidence for each candidate) and increase bandwidth requirements (to transfer long lists of candidates from database machines to be tallied).

In this paper, we propose a method based on *permutation-grouping*. Permutation grouping addresses the problem of non-distinctive hash functions selected in LSH, by observing and adjusting for the underlying structure in the data. We skew the distribution from which the hash functions are sampled, and intelligently select their grouping, to ensure that the resulting L keys are as distinctive as possible. Our results show that our grouping method maintains the attractive statistical properties of LSH, while considerably reducing the retrieval cost.

2. FAST MATCHING WITH MINHASH + LSH

The goal of our matching system is to be robust to the types of degradations that we expect to see between database entries and probes. In the audio domain, the system is designed to handle random noise, competing structured noise (other songs in the background, voices), echoes, poor mp3 encoding, playback over cell phones, etc. In the visual domain, we address common variations seen in images – poor jpeg encoding, changes in aspect ratio, saturation and hue, overlayed text, sharpening and blurring, etc.

The matching system works in the following basic steps. To create the reference database, Haar-Wavelets of the image (or spectrogram segment) are first computed. By itself, the wavelet-image is not resistant to noise or degradations. To reduce the effects of noise, while maintaining the major characteristics of the image, we select the t top wavelets (by magnitude) and discard the rest. Jacobs [6] further determined that after keeping only the top wavelets, the coefficient magnitudes are not needed for

effective retrieval: instead the sign bits alone could be used. As memory usage is a primary concern in this system, this same top-wavelet-sign representation is used here. The sparsity of the resulting top-wavelet vector makes it amenable to further reduction using the MinHash [4].

MinHash works with sparse binary vectors as follows: Select a random, but known, reordering of all the vector positions. For each vector permutation, measure in which position the first '1' occurs; this projection is the first component of the signature. Note that for two vectors, *v1 & v2*, the probability that first_1_occurrence(*v1*) = first_1_occurrence(*v2*) is the same as the probability of finding a row that has a 1 in both *v1* and *v2,* from the set of rows that have 1 in either *v1* or *v2*. Therefore, for a given permutation, the MinHash values match for *v1* and *v2* if the first position with a 1 is the same in both bit vectors, and they disagree if the first such position is a row where one but not both, vectors contained a 1. *Note that this is exactly what is required; it measures the similarity of the sparse bit vectors based on matching "on" positions.* A full signature is the concatenation of *M* MinHash projections. These *M* projections are then placed into the *L* LSH tables by selecting *M/L* permutations for each hash key.

For retrieval, each probe is hashed into the *L* tables in the same manner. The best match from the set of candidate neighbors (the union of the entries found in the *L* matching bins—one from each table) is the one that matched in the most hash-tables.

Given this formulation, we created a full system using the following parameters[1]. For images, each image to be inserted into the database was reduced to a 32×32 thumbnail, the Haar wavelets were extracted independently for 5 channels of the image (R,G, B, I, Q) and the signs of the top-50 wavelets by magnitude were kept; others were set to 0. For the audio spectrogram images, which are based on those created in [7][8], a single channel can be used. The length of the subfingerprints (the number of MinHashes that were concatenated for use as the key) into each hash table was varied from 3-8 and we used L=10; therefore, for each channel the total length of the signature varied from 30 (3×10) MinHashes to 80 (8×10). Each table had 10^6 bins.

We measure the retrieval accuracy of the system in the standard manner, by examining the percentage of probe queries (consisting of severely degraded probes) that found the original entry in the database. In live deployment, we must plan for excessive peak-time query loads (lookups for matching signatures); because the queries will be farmed out to multiple machines (10s-100s), the number of elements returned for each lookup must be kept small. Large numbers of matching elements returned will not only incur large computational penalties when the tallies are maintained to determine which image has the most votes, but will also incur large amounts of bandwidth as large lists are transferred between machines. To keep the number of candidates small, we examine two metrics; the first provides insights into system performance, the second provides the exact measurement we need to minimize.

(1) *Max-Occupancy:* the max number of elements in a bin for each table, averaged over all hash tables; the lower this is, the lower the max bandwidth will be.

(2) *Total-Elements-Returned*: average number of total elements returned for a lookup of a probe (across all hash tables and all channels).

For the baseline tests, shown in Table I, image probes were created by random combinations of added noise, auto-color "enhancement", overlaid large text, blurring, sharpening, ±contrast, ±saturation, and aspect ratio modification.

Table I: System with 2×10^6 images in DB, 14,100 probes. L=10 Hash Tables per Channel, 10^6 elements in each table.

MinHashes	Correct	Max Occupancy of a Bin	Total elements Returned
3 per key	99.1%	107,026	432,193
4 per key	99.0%	45,013	91,908
5 per key	98.9%	21,172	19,939
6 per key	98.6%	13,904	4,996
7 per key	98.2%	11,481	1,363
8 per key	97.6%	10,122	476

As can be seen, the number of total elements returned drops dramatically as the size of the hash-key increases; the longer the hash-key, the better the distribution across the bins of the hash tables. This is corroborated by the fact that the maximum occupancy of any bin also falls dramatically. However, the drawback of increasing the hash-key size is that it *increases* the threshold for finding a match. In order for the probe to be hashed to the same bin as the correct match, it must be an exact match for a larger set of keys; therefore, less degradation is tolerated.

In order to maintain at least a 99% retrieval rate, we can use at most 4-5 MinHashes per key. As can be seen, however, for these settings, there are a large number of keys that are returned for each lookup. Note that if the entries had been distributed perfectly across the hash table, per channel, each bin would hold only 2 elements (therefore, ideally we would examine only 20 elements per channel (2 × 10 hash tables per channel). However, we are far from this number. Next, we explain why this clumping happens in some hash bins, and demonstrate how to reduce its effects.

3. PERMUTATION GROUPING

In this section, we first describe the cause of the uneven clumping in the hash bins that led to the large number of element returned per lookup. Second, we propose a method, *permutation grouping*, to avoid the clumping.

[1] These parameters were tuned through significant experimental testing; see [1] for a complete description of parameters interactions and full details on the audio spectrogram settings.

The first insight into the cause of the problem is found in the distribution of the MinHash signatures. Recall that the MinHash signature measures the first 'on' position in a random permutation of a sparse vector. When each element of the original sparse vector has an independent and identical (i.i.d.) probability of being on, the MinHash signatures exhibit a smooth drop in probability as the positions get larger. Using this model with p for the 'on' probability, the MinHash output space will follow a geometric distribution: $P(\text{reference}=n) = p(1-p)^n$; i.e. there are n 'off' entries before there is a 'on' entry. This distribution outputs the lowest values with the highest frequency, in a monotonic decreasing distribution. In Figure 2A, we see that the generated distribution (for i.i.d data) is almost exactly the expected; the entropy is 6.7.

In contrast, in Figure 2B, we look at 10 sample permutations and examine the probability of occurrence for each position; it is clearly non-uniform and severe clumping of the samples is apparent. For these distributions, the entropy is approximately 4.0; significantly lower than with i.i.d. samples. The entropy distributions of 100 sample permutations of i.i.d. and top-wavelets data is shown in Figure 3. Importantly, with an entropy of 4, only 16 of the 255 positions are being effectively used; with the i.i.d. samples (entropy=6.7), approximately 105 positions are.

Given the large variation in the entropies observed by the random selection of permutations with real data (Figure 3); the fist step in intelligently designing the L hash functions used for LSH is to use those permutations with highest entropies. However, recall that when the MinHash signatures are used with LSH, multiple MinHash projections are concatenated to form a single hash-key (in the previous experiments, the groups consisted of 3-8 elements). Rather than simply selecting high entropy permutations to place

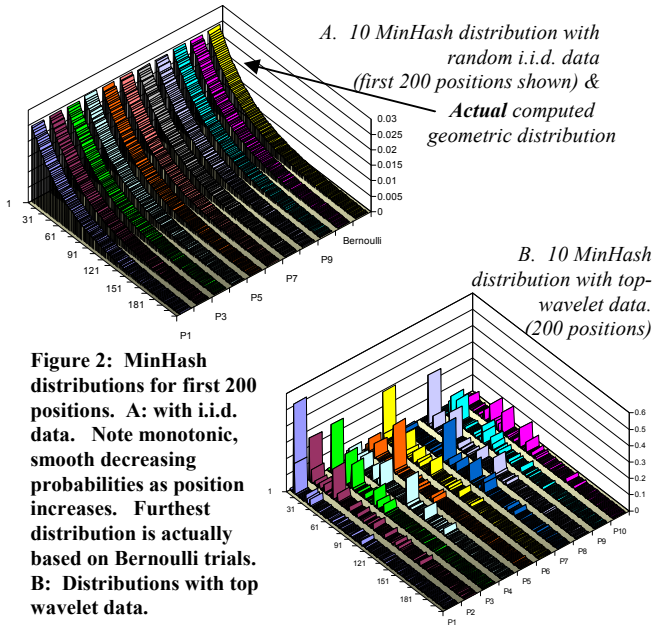

Figure 2: MinHash distributions for first 200 positions. A: with i.i.d. data. Note monotonic, smooth decreasing probabilities as position increases. Furthest distribution is actually based on Bernoulli trials. B: Distributions with top wavelet data.

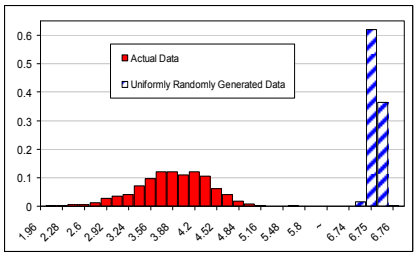

Figure 3: (-1.0*) Entropy of 100 MinHash permutations using real data (left) and i.i.d. data (right, striped). Note the large differences in averages (4.0 vs. 6.7). X-Axis Scale exaggerated on right.

together, a more principled method is to use the Mutual Information (MI) between permutations to guide which permutations are grouped. Mutual information is a measure of how much knowing the value of one variable reduces the uncertainty in another variable. Formally, in terms of entropy, mutual information is defined as:

$$I(X;Y) = H(X) - H(X|Y) = H(X) + H(Y) - H(X,Y)$$
$$= \sum_{y\in Y}\sum_{x\in X} p(x,y)\log\frac{p(x,y)}{p(x)p(y)}$$

To determine whether there is sufficient mutual information variance to use this as a valid signal, for 100 permutations, we examined the mutual information between all pairs (100*99/2 samples). The results are shown in Figure 4 for i.i.d. and real data. Although the same general shape, *note the significantly longer tail for real data.* The existence of this tail is important; if the permutations with high mutual information are in the same group, clumping will be increased (intuitively, since the new permutations will be correlated, the bits used for that hash will be inefficiently used, and the spread of the items in the bins will diminish).

In order to create groups of low mutual information permutations to put together into hashing chunks, we use a greedy selection procedure that is loosely based on the algorithm used in Chow and Liu [3]. Whereas [3] created a spanning tree that *maximized* the MI between sets of variables, we use a similar greedy selection procedure to *minimize* the mutual information in order to create a forest of trees; each of whose constituents are the set of permutations that are grouped together.

First, for all of the L groups of hashes, an initial permutation is assigned. These are chosen to be the L permutations with the highest unconditional entropy. These L are added into the selected set, S. Using G as the set of L groups, and B as the size of the group (number of MinHashes per key), the remaining permutations are selected iteratively through one of the three procedures:

(1): $\min_{s\notin S, g\in G, s.t.|g|<B}\left(\min_{t\in g}\left(I(s,t)\right)\right)$: Find the unselected permutation, s, with the minimum MI with any of member of a group that does not already have B members. Once found, add s to the group g, ($t \in g$) and add s to S. This is the most aggressive of the three methods as it uses the lowest MI to a *single* member of the group to make the next selection.

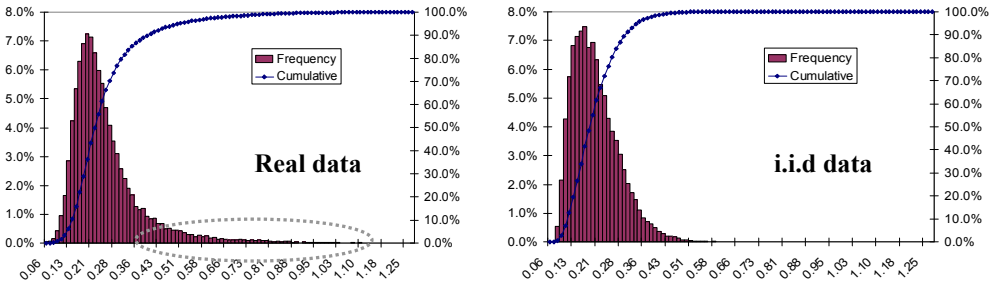

Figure 4: Histogram of Mutual information between all pairs of 100 MinHash permutations using real data (left) and i.i.d. data (right). Line is cumulative probability. Note the longer tail observed with real data (circled). Randomly chosen, unlucky, combinations will yield clumping.

(2): $\min_{s\notin S, g\in G, s.t. |g|<B}\left(\sum_{t\in g} I(s,t)\right)$: Find the unselected permutation as above, except assign the MI to the group as the MI of the candidate *summed across* the members already in the group.

(3): $\min_{s\notin S, g\in G, s.t. |g|<B}\left(\max_{t\in g}(I(s,t))\right)$: Find the unselected permutation as above, except assign the MI to the group to be the *maximum* of the MIs between s and any member of that group. Select the minimum of these across groups and unselected s. This is the most conservative of the procedures; it minimizes the worst of the correlations.

Note that many more permutations can be generated than need to be used; this allows us to generate a large pool from which to select. These procedures run in $O(n^2)$ time, where n is the # of permutations. Importantly, this computational load is only incurred during system design, not during matching, retrieval, or database generation.

4. EXPERIMENTS

For the experiments, the trials described in Section 2 were rerun. In these experiments, however, instead of randomly grouping the permutations, they were grouped in the 3 manners described above. A total of 100 permutations were generated, from which 30-80 were selected (depending on the experiment, as shown in Table 2).

The findings all revealed dramatically improved results, in terms of the maximum occupancy of any bin, and the total elements-returned. There was no significant change in the number of correct matches. Due to space restrictions, we show the results, in Table 2, for only method #3 described above; this had the best overall performance. From Table 2: the maximum occupancy of any bin in the hash tables has dramatically dropped for all MinHash settings. The maximum drop was 46% (when 4 hashes per key were employed); the minimum, 14% when 7 hashes were employed). The more pronounced effect with the smaller number of keys occurs because, as the number of keys increases, the effect of a few 'unlucky' permutation combinations diminishes. Most importantly, the total number of elements returned has decreased between 30% and 51%. This yields not only substantial savings in the amount of computation required to tabulate and track the candidates, but also eases the enormous network burden caused by transferring large lists of candidates between multiple machines.

Table 2: System with 2×10^6 images in DB, 14,100 probes. L=10 Hash Tables per Channel, 10^6 elements in each table. *% improvement over not using MI-based grouping also shown.*

MinHashes	Correct	Max Occupancy		Total Elements	
3 per key	99.1%	63,273	***-41%***	302,605	***-30%***
4 per key	99.0%	24,195	***-46%***	55,978	***-39%***
5 per key	98.8%	13,762	***-35%***	10,994	***-45%***
6 per key	98.5%	10,212	***-27%***	2,463	***-51%***
7 per key	98.0%	9,852	***-14%***	684	***-50%***
8 per key	97.2%	8,316	***-18%***	243	***-49%***

5. CONCLUSIONS & FUTURE WORK

With no extra computation cost during retrieval time, and with no significant change in retrieval accuracy, we were able to significantly reduce the number of candidates (by 30-50%) that need to be examined. We achieved this by better selecting the permutations that were grouped together for hashing; this minimized their MI and more effectively used the bits. This has a large benefit in the context of large systems; the fewer the candidates, the better the computation and bandwidth performance.

The performance improvement was demonstrated across all sizes of hash keys examined. In our implementation, we will use between 4-6 hashes per group in live systems; thereby resulting in savings of over 40%. In the future, we would like to examine directly changing the distribution of the hashes by augmenting the MinHash permutation scheme.

6. REFERENCES

[1] Baluja, S., Covell, M. "Audio Fingerprinting: Combining Computer Vision & Data Stream Processing", *ICASSP-2007*.

[2] Casey, M., Slaney, M. (2006) Song intersection by Approximate Nearest Neighbor Search, *ISMIR 2006*.

[3] Chow, C., Liu, C, "Approximating Discrete Probability Distributions with Dependence Trees*", IEEE-Info Theory* 14(3)

[4] Cohen, E. et. al (2001) Finding interesting associations without support pruning. *Knowledge and Data Engineering*, 13(1):64–78.

[5] Gionis, A., P. Indyk, R. Motwani (1999), Similarity search in high dimensions via hashing. in *Proc. VLDB*, pp. 518–529.

[6] Jacobs, C., Finkelstein, A., Salesin, D. (1995) Fast Multiresolution Image Querying. *in Proc of SIGGRAPH 95*.

[7] Ke, Y., D. Hoiem, R. Sukthankar (2005). Computer Vision for Music Identification. In *CVPR* pp. 597-604.

[8] Haitsma & Kalker, "A Highly Robust Audio Fingerprinting System", ISMIR-2002.