A Comparison of Extended Fingerprint Hashing and Locality Sensitive Hashing for Binary Audio Fingerprints

Kimberly Moravec
Department of Computer Science
University College London
Malet Place, London WC1E 6BT, UK
k.moravec@cs.ucl.ac.uk

Ingemar J. Cox
Department of Computer Science
University College London
Malet Place, London WC1E 6BT, UK
ingemar@cs.ucl.ac.uk

ABSTRACT

Hash tables have been proposed for the indexing of high-dimensional binary vectors, specifically for the identification of media by fingerprints. In this paper we develop a new model to predict the performance of a hash-based method (Fingerprint Hashing) under varying levels of noise. We show that by the adjustment of two parameters, robustness to a higher level of noise is achieved. We extend Fingerprint Hashing to a multi-table range search (Extended Fingerprint Hashing) and show this approach also increases robustness to noise. We then show the relationship between Extended Fingerprint Hashing and Locality Sensitive Hashing and investigate design choices for dealing with higher noise levels. If index size must be held constant, the Extended Fingerprint Hash is a superior method. We also show that to achieve similar performance at a given level of noise a Locality Sensitive Hash requires nearly a six-fold increase in index size which is likely to be impractical for many applications.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Indexing methods*

General Terms

Algorithms,Theory,Performance

1. INTRODUCTION

Identifying digital music files, e.g. mp3, is an essential task of various applications. In particular, when importing mp3 files, music management applications such as iTunes need to identify the title of the song, the artist, etc. in order to enter the file into their database. This identification is typically performed by extracting one or more digital fingerprints from the song and comparing the fingerprint against a database of known fingerprints. Identification is also needed for applications such as tracking illegal use of copyrighted media, or for the automatic monitoring of broadcast media for the purposes of royalty collection and advertisement verification [1].

The problem of music identification can be broadly divided into two sub-problems. The first relates to feature extraction, i.e. the choice of features used to form the fingerprint. The second relates to nearest neighbor search, i.e. how to efficiently find the closest match to the extracted fingerprint.

This paper concentrates on the nearest-neighbor search. We note that the extracted fingerprint is often a high-dimensional binary vector, e.g. of dimension 8,000, corrupted by compression, transmission and other processes. We assume that a good fingerprint generation scheme results in an approximately uniform distribution of fingerprints, and that the correct answer to a query is likely within a certain range which can be determined empirically. (A review of fingerprint generation methods can be found in [5].)

Nearest neighbor search is a well-studied problem [8]. Early on it was recognized that algorithms which worked well for low dimensions did not scale to higher dimensions [8]. Techniques have been developed for higher dimensions (to about twenty) [3, 17, 7]; other work has focused on developing approximate methods for even higher dimensions [12, 14, 2, 16].

For music identification, one common search algorithm is based on hashing subfingerprints [9]. Section 2 describes three hash-based methods, fingerprint hashing and its derivative extended fingerprint hashing, and the Locality Sensitive Hash–based sample hash. We derive equations for the search success rate as a function of bit error rate for each method. In Section 3 we model search speed as a function of the total number of retrieved results, and show that modifying the parameters of a sample hash, one can achieve a lower search cost than an extended fingerprint hash, at the cost of a larger index. The analysis in this paper links two indexing methods under a common framework, one which allows designers to implement binary database indexes in a principled way.

2. HASH-BASED SEARCH

In this section we describe a basic fingerprint hash and show that the successful search rate is a function of fingerprint length, hash-table key length and the underlying bit error rate of the queries. We then describe an extended fingerprint hash based on a range search and derive its corresponding successful search rate. We also describe the Locality Sensitive Hash–based sample hash, and show that there

ICMR '11, April 17-20, Trento, Italy

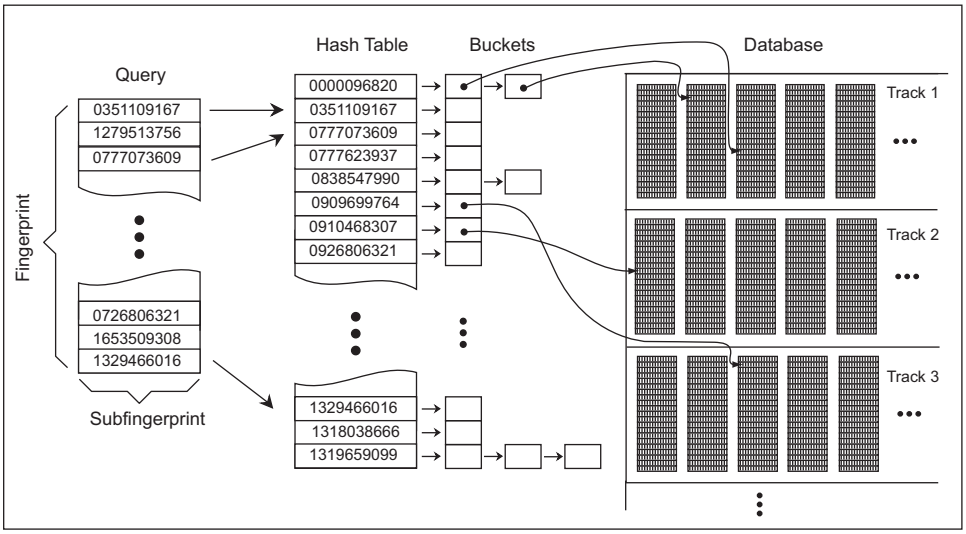

Figure 1: A diagram of fingerprint hashing for audio fingerprints.

is a link between sample hashing and extended fingerprint hashing.

2.1 Fingerprint Hash

We briefly describe the fundamentals of hash-based search for music identification, based on the work of Haitsma and Kalker [9]. To distinguish between this method and other hashed based methods in this paper, we denote this method *Fingerprint Hash*.

In [9] each fingerprint consists of 8192 bits. This fingerprint is disjointly sub-divided into 256 subfingerprints of 32 bits. Using the subfingerprints as keys, a hash table is constructed. There are 2^{32} possible subfingerprints, but only those that exist in the database are entered in the hash table. Buckets for each key contain pointers to songs that are known to contain this subfingerprint. A single subfingerprint can occur at many locations within a single song and can also appear in multiple songs. This hash table is analogous to an inverted index [19, 15] used in text retrieval. As such, this hash-based method is sometimes referred to as an inverted file index [5]. The data structure is illustrated in Figure 1.

For a query, each subfingerprint is looked up in the hash table, and a list of the candidate matching fingerprints is generated. If more than one candidate exists, the Hamming distances between the query and all candidate fingerprints are computed, and the closest candidate is chosen.

The evaluation of an indexing and search system has multiple components: memory needed, compute cost, the effect of false positives and false negatives. A fingerprint hash search requires multiple retrievals, each of which may return multiple values. The retrieval of a hash table value is of $O(1)$, so we expect the bulk of the cost of a query will due to the costly high-dimesional comparison between the query and each candidate fingerprint retrieved. Thus we model the cost of a query as the number of candidate fingerprints retrieved per search. Another major factor we consider in this paper is the storage size of the hash table index.

The hash table system described above works well as long as the bit error is low (less than about 15%). This because the system assumes that *at least one subfingerprint is error free*. For bit error rates of less than 15%, this is very likely.

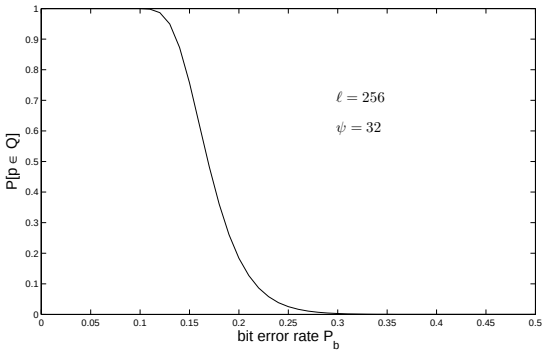

Figure 2: The probability, $P[p \in Q]$, that a fingerprint contains at least one error-free subfingerprint as a function of the bit error rate, P_b

Consider a fingerprint query q corrupted by noise, and let p be the correct match in the database. Querying the database will return a set Q of candidate fingerprints, the contents of all the buckets accessed by each subfingerprint key. If one or more subfingerprints of p and q match, then $p \in Q$ and p can be found by considering each member of Q.

Let P_b be the probability that a single bit is in error; this is our underlying *bit error rate*. Then the probability that an entire subfingerprint of length ψ is error free is $(1-P_b)^\psi$. The probability of at least one error in a subfingerprint is given by $1-(1-P_b)^\psi$. If a full fingerprint consists of ℓ subfingerprints, then the probability that all ℓ subfingerprints have errors is given by $(1-(1-P_b)^\psi)^\ell$. The probability that a fingerprint has at least one error-free subfingerprint and will therefore return the correct match p in the candidate fingerprint list Q is therefore

$$P[p \in Q] = 1 - (1 - (1 - P_b)^\psi)^\ell \quad (1)$$

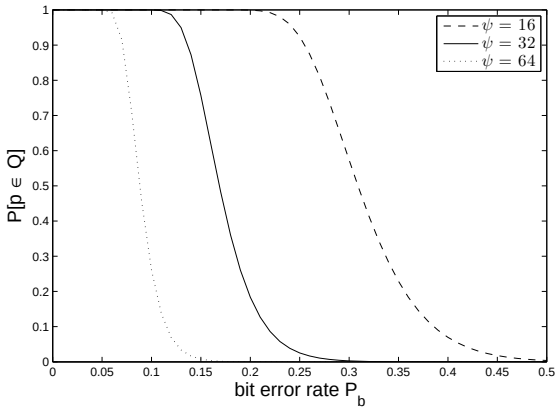

Figure 3: Fingerprint Hash: The probability of a successful search, $P[p \in Q]$, for subfingerprint lengths, $\psi = 16$, 32 and 64.

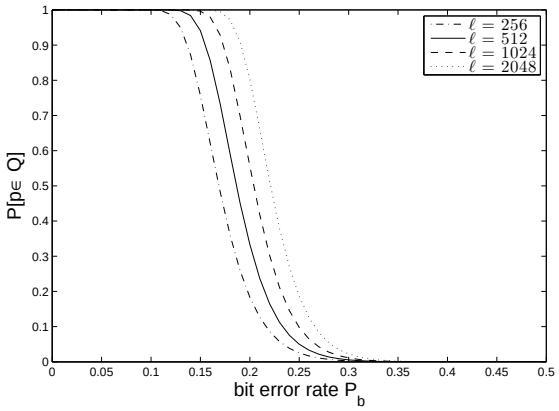

Figure 4: Fingerprint Hash: The probability of a successful search, $P[p \in Q]$ for the number of subfingerprints in a fingerprint, $\ell = 256$, 512, 1024, and 2048.

This probability is plotted in Figure 2 for the case where the number of subfingerprints $\ell = 256$ and each subfingerprint consists of $\psi = 32$ bits.

As one might expect, the probability that a fingerprint contains at least one error-free subfingerprint is high for low bit errors and low for high bit errors, with a rapid transition at a bit error rate of about 0.175. If we require a successful search rate of 90%, this system will not work for bit errors greater than about 0.136. This corresponds with empirical observations that this method does not work well for higher bit errors [9, 5]. In general, Equation 1 allows us to predict the performance of a fingerprint hash for any bit error rate. Such an analysis has not been presented before. Furthermore, Equation 1 identifies two parameters, the length of a subfingerptint, ψ, and the number of subfingerprints, ℓ, which can be altered to improve the robustness of the basic hash-based search to higher bit error rates as required.

The effect of various subfingerprint lengths ψ is shown in Figure 3. Decreasing ψ leads to a greater bit error rate P_b for which $P[p \in Q]$ is high. The search success rate $P[p \in Q]$ is maximized when $\psi = 1$ but it results in a hash table of two keys, '1' and '0', and the search becomes exhaustive. In practice, ψ is constrained on the one hand to be sufficiently large that the hash table contains only a few entries in each bucket, and on the other hand ψ must be sufficiently small to ensure that for a given bit error rate, the probability is high that at least one subfingerprint is error-free.

The effect of various fingerprint lengths ℓ is shown in Figure 4. Increasing ℓ leads to a greater bit error rate P_b for which $P[p \in Q]$ is high (although the improvement is much smaller than corresponding changes to ψ). In this case, the limiting factor is the length of the fingerprint. Increasing ℓ increases the dimensionality of the fingerprint, and dimensionality is directly proportional to the time needed to capture a fingerprint query. Moreover, increasing ℓ also increases the number of candidate subfingerprints that must be examined, as discussed next.

Both of these methods increase robustness to higher bit error rate P_b at the cost of returning more results, that is by increasing the size of Q. Let n be the size of the database in fingerprints, τ be the number of hash table accesses, and b be the average number of items in a bucket (also known as the Load Factor in conventional hashing). Then the number of results returned by the search, $|Q|$ can be modeled by

$$|Q| = \tau b \tag{2}$$

For a basic fingerprint hash as in [9], hash table accesses $\tau = \ell$ and the average number of fingerprints in a bucket $b = \frac{n\ell}{2^\psi}$, leading to the number of candidate fingerprints returned as

$$|Q_{FH}| = \frac{n\ell^2}{2^\psi} \tag{3}$$

The number of candidate fingerprints returned scales as ℓ^2, but we can reduce this to ℓ by using multiple hash tables, that is ℓ tables, one per subfingerprint [1]. The average number of items in a bucket then becomes $b_{FH} = \frac{n}{2^\psi}$ and Equation 2 becomes

$$|Q_{FH}| = \frac{n\ell}{2^\psi} \tag{4}$$

Irrespective of whether single or multiple hash tables are used, the size of the fingerprint hash index is of $O(n\ell)$.

2.2 Extended Fingerprint Hash

As bit error rate increases, the assumption no longer holds that at least one subfingerprint is error free. However, it may still be true that *at least one subfingerprint has only one error*. In this case, for a fingerprint hash to succeed, it needs to search on the original subfingerprint key q, and all keys within a Hamming distance 1 of q. For even higher error rates, the approach can be generalized to assuming *at least one subfingerprint has no more than r errors* and success is achieved by exhaustively searching all possible subfingerprint keys within a Hamming distance r of q. For binary subfingerprints, these keys can be trivially generated by systematically flipping bits.

This approach, which we will call *Extended Fingerprint Hashing*, was proposed in [9]. The probability of a correct match is increased; Let e be the number of errors in a subfingerprint, and the probability that e is less than r be $P[e \leq r]$.

[1]We implement this in practice by adding a positional prefix to the subfingerprint key.

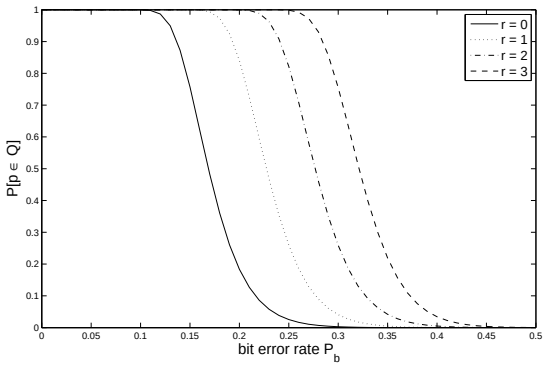

Figure 5: The probability of a successful search, $P[p \in Q]$, as a function of bit error , P_b, for extended fingerprint hash search values of r=0, 1, 2, and 3.

Then:

$$P[e \leq r] = \sum_{i=0}^{r} \binom{\psi}{i} P_b^i (1 - P_b)^{\psi - i} \tag{5}$$

and hence:

$$P[p \in Q] = 1 - (1 - P[e \leq r])^{\ell} \tag{6}$$

The increased robustness of searching over $r = 0$ to 3 is shown in Figure 5. We observer that for a fixed probability of a successful search, $P[p \in Q] = 0.9$, the effective bit error rates are $P_b = 0.1364, 0.1930, 0.2425, 0.2840$ for $r = 0, 1, 2, 3$ respectively.

There is a cost; for a fingerprint hash, the number of keys to be searched per query fingerprint is ℓ. In contrast, an extended fingerprint hash search must search the hash table

$$\tau_{EF} = \ell \left(1 + \binom{\psi}{r}\right) = \ell \left(1 + \frac{\psi!}{r!(\psi - r)!}\right) \tag{7}$$

times. The equation for the total cost of the search, $|Q_{EF}|$, is given by:

$$|Q_{EF}| = \frac{n\ell}{2^{\psi}} \left(1 + \binom{\psi}{r}\right) \tag{8}$$

This equation, larger than the cost of a fingerprint hash (Equation 4) by a factorial term, increases quickly for higher r, but is unrelated to database size. This may make it a realistic option for larger databases, as opposed to other methods with search costs that increase with database size, such as metric trees [18]. For instance, on a database of $n = 10^{10}$ fingerprints, an extended fingerprint hash searches 0.0014% of the database at a range of $r = 2$ and 0.03% of the database at a range of $r = 3$.

We note that the size of an extended fingerprint index is the same as that for a basic fingerprint hash index, and is of $O(n\ell)$.

The predictions of successful searches as a function of bit error rate derived in Equations 1 and 6 for an extended fingerprint hash were tested on a database of 100,000 synthetically generated fingerprints, uniformly distributed over the 8192-dimensional Hamming hypercube. The fingerprint hash method detailed in uses parameters $\psi = 32$, $\ell = 256$

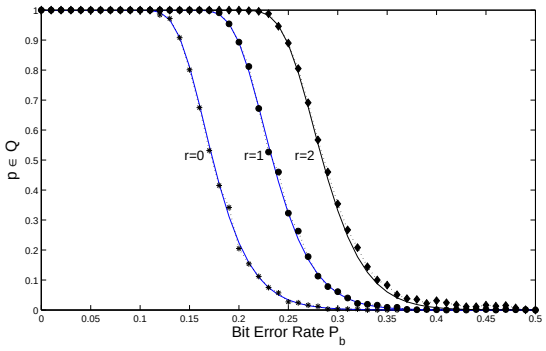

Figure 6: Experimental results on synthetically generated database of 100,000 uniformly distributed points. Solid lines indicate theoretical predictions, starred lines indicate experimental results. Results are averaged over 1000 queries.

Database Size	range searched		
	r=0	r=1	r=2
100k	0.0004%	0.0007%	0.0035%

Table 1: The average number of unique results returned by an extended fingerprint hash as a percentage of the database size.

and r ranging from 0 to 3. Figure 6 shows the fraction of fingerprints correctly found for increasing bit error. The solid lines are the performance predicted by our model, and the starred lines are the experimental results. The results are averaged over 1000 queries. As can be seen from the figure, the model predicts the performance of the inverted file index for uniformly distributed data very closely.

For each range r, the number of unique results returned as a percent of the total database, is given in Table 1. These numbers are below what would be predicted by our analysis, as entries for many of the keys generated by extended fingerprint hashing do not exist for a small database.

2.3 Sample Hash (Locality Sensitive Hash)

Various researchers have independently proposed hash-table based structures for the indexing of binary data [4, 11, 13, 6, 10]. Perhaps the best-known method of these is *Locality Sensitive Hashing* (LSH) [10, 2].

The LSH approach defines a family $\mathcal{H}$ of *locality sensitive* hashing functions h mapping $\mathbb{R}^d$ to a universe U such that the probability that $h(p) = h(q)$ is related to the Hamming distance $d(p, q)$ as follows:

- if $d(p, q) \leq R$ then $P_{\mathcal{H}}[h(p) = h(q)] \geq P_1$
- if $d(p, q) \geq cR$ then $P_{\mathcal{H}}[h(p) = h(q)] \leq P_2$
- $P_1 > P_2$

where c is a number greater than one.

A basic family of hash functions is the family

$$\mathcal{H} = \{\forall h_i(p) | h_i(p) = p_i \text{ and } 1 \leq i \leq d\}$$

where p_i denotes the bit value of p at index i. This function

maps $\{0,1\}^d$ to $\{0,1\}$. Then

$$\left(1-\frac{R}{d}\right) = P_{\mathcal{H}}[h(q)=h(p)] \geq P_1 \tag{9}$$

and if $d(p,q)=cR$ then $P_2=\left(1-\frac{cR}{d}\right)$, satisfying the LSH condition $P_1 > P_2$.

Another LSH family of hash functions is $\mathcal{G}$, where

$$g_j(q) = \{h_{i_1}(p), \ldots, h_{i_k}(p)\}$$

where the indexes i_1 through i_k are drawn randomly and uniformly from d . This is equivalent to *randomly sampling* k bits from p. We note that

$$\begin{aligned} P_1 &= P_{\mathcal{G}}[g(p)=g(q)] &(10)\\ &= \prod_{i=1}^{k} P[q_i=p_i] &(11)\\ &= \left(1-\frac{R}{d}\right)^k &(12)\end{aligned}$$

and likewise if $d(p,q)=cR$ then $P_2=\left(1-\frac{cR}{d}\right)^k$. If $d \geq k \geq 1$ and $c>1$, the LSH condition $P_1 > P_2$ is satisfied. A set consisting of L of these randomly chosen functions form the keys to the hash table (with a separate hash table for each $g_i(p)$) a method which we will call a *Sample Hash*.

Sample hashing accesses the hash table $\tau_{SH}=L$ times and the average length of a bucket is $b_{SH}=\frac{n}{2^k}$ giving the size of the results list returned as:

$$|Q_{SH}| = \frac{nL}{2^k} \tag{13}$$

The size of the sample hash index is $O(nL)$.

We show here that fingerprint hash is also an LSH hash. Fingerprint hash forms the keys to the hash table by partitioning the fingerprint vector p into ℓ subfingerprints of length ψ,

$$\begin{aligned}\mathcal{F} = \{&f_1(p) = \{p_1, \ldots, p_\psi\},\\ &f_2(p) = \{p_{\psi+1}, \ldots, p_{2\psi}\},\\ &\ldots,\\ &f_\ell(p) = \{p_{(\ell-1)\psi+1}, \ldots, p_{\ell\psi}\}\}\end{aligned}$$

In this case, analogously to the case above, $(1-R/d)^\psi = P_{\mathcal{F}}[f(q)=f(p)] \geq P_1^\psi$, and the LSH properties follow.

Sample Hash accesses a hash table L times, with different randomly generated $g(q)$ functions. Fingerprint hash accesses the table ℓ times, where $\ell=\frac{d}{\psi}$. The probability of a successful Sample Hash search is of the same form as Equation 1.

$$1-(1-(1-P_b)^k)^L \tag{14}$$

A sample hash forms its hash keys by *randomly sampling* the fingerprint. In contrast, a fingerprint hash *partitions* the fingerprint into disjoint subsets. Assuming the bits of the vector p are independent and identically distributed[2], it does not matter which bits are chosen nor in which order they are chosen to make up the hash table keys. A fingerprint hash and a sample hash with $\psi=k$ and $\ell=L$ will perform similarly.

[2]In practice, this may not entirely be true, but a good fingerprint system's bits will approach independent and identical distribution.

	Probability of a correct search $P[p \in Q]$	Hash Table Accesses τ	Hash Table Storage
Fing. Hash	$1-(1-(1-P_1)^\psi)^\ell$	ℓ	$O(n\ell)$
Ext.Fing. Hash	$1-(1-P[e \leq r])^\ell$	$\ell\left(1+\binom{\psi}{r}\right)$	$O(n\ell)$
Sample Hash	$1-(1-(1-P_1)^k)^L$	L	$O(nL)$

Table 2: Three Hash Methods Compared

One advantage of a sample hash is that k and L are not determined by fingerprint dimension d. One is free to increase L to very high values without needing to increase the dimension of the vector p, although eventually the same bits will be sampled multiple times. The equivalent equations for the probability of a successful search, the number of hash table accesses and the size of the index for each method are given in Table 2.

3. THE COST OF SEARCHING

This paper has so far detailed two general approaches for increasing the robustness of hash tables to higher query bit errors: i)modifying ψ and ℓ (or analogously k and L) and ii) exhaustively searching within a Hamming distance r. For each of these methods there is an increase in the number of data items retrieved, and a corresponding decrease in the performance of the algorithm, as each retrieved data item must undergo further processing to determine the correct match. In this section we define four methods and look at the relative increase in $|Q|$ for each one.

The four methods compared are:

- **Method I:** Extended Fingerprint Hash (Baseline)
- **Method II:** Sample Hash, modify k only
- **Method III:** Sample Hash, modify L only
- **Method IV:** Sample Hash, modify k and L

In practice, it is not necessary for $P[p \in Q]$ to equal one, i.e. applications can accept some (small) level of failure. Of course, an acceptable failure rate will depend on the application. For example, a song identification system may accept a higher error rate than an advertisement identification system. This is because, in the former case, the misidentification simply results in an incorrect entry in say the iTunes database, while in the latter case, the misidentification may result in the broadcaster being wrongly accused of failing to air an advertisement. We choose a reasonable precision value of 90%. In the following discussion we assume the number of fingerprints n in the database to be 10^{10}.

From Figure 5, it can be seen that a fingerprint hash or a sample hash can handle a P_0 of 0.1364 when $P[p \in Q]=0.90$ (for parameters $\psi=k_0=32$ and $\ell=L_0=256$). The first column of Table 3 shows the higher bit error rate, P_b, that can achieve $P[p \in Q]=0.90$ for an extended fingerprint hash by flipping r bits. This gives a baseline for comparing Methods I through IV.

P_b	r	τ	b
0.1364	0	256	2.33
0.1930	1	8,448	2.33
0.2425	2	135,424	2.33
0.2840	3	2,810,112	2.33

Table 3: Method I: Extended Fingerprint Hash. The range over which the method searches is r and P_b is the corresponding bit error rate for which the probability of a successful search is 90%. The number of hash-table accesses is τ and the average number of fingerprints in a bucket is b.

P_b	k_1	τ	b
0.1364	32	256	2.33
0.1930	17	256	76,294
0.2425	13	256	$1.2 \cdot 10^6$
0.2840	11	256	$4.9 \cdot 10^6$

Table 4: Method II: Sample Hash, modifying k only. P_b is the bit error rate for which the probability of a successful search is 90%, and k_1 is the length of sample needed to achieve this, given the number of samples L=256. The number of hash-table accesses is τ and the average number of fingerprints in a bucket is b.

P_b	L_1	τ	b
0.1364	256	256	2.33
0.1930	20,500	20,500	2.33
0.2425	295,600	295,600	2.33
0.2840	1,998,500	1,998,500	2.33

Table 5: Method III: Sample Hash, modifying L only. P_b is the bit error rate for which the probability of a successful search is 90%, and L_1 is the number of samples needed to achieve this, given a sample size of k=32. The number of hash-table accesses is τ and the average number of fingerprints in a bucket is b.

P_b	k_1	L_1	τ	b
0.1364	31	221	221	4.66
0.1930	31	1,815	1,815	4.66
0.2425	31	12,922	12,922	4.66
0.2840	31	74,119	74,119	4.66

Table 6: Method IV: Sample Hash, modifying k and L. P_b is the bit error rate for which the probability of a successful search is 90%, and L_1 is the number of samples needed to achieve this, given a smaller sample size of k=31. The number of hash-table accesses is τ and the average number of fingerprints in a bucket is b

3.1 Method I–Extended Fingerprint Hash

Table 3 shows the numerical values of hash table accesses τ and average bucket size b resulting for an extended fingerprint hash of a range search $r = 0$ to 3. We did not consider values of $r > 3$ because the number of candidate matches becomes impractical.

Let $|Q_0|$ be the size of the candidate results list from a basic fingerprint hash, and let $|Q_1|$ be the size of the candidate results list from an extended fingerprint hash of range r. From Equations 4 and 8 it can be seen that the relative increase in the size of the results list for Method I is in general:

$$\frac{|Q_1|}{|Q_0|} = \left(1 + \binom{\psi}{r}\right) \tag{15}$$

For the particular values in Table 3, going from $r = 0$ to $r = 1$ is a 33-fold increase in the number of candidate results returned.

3.2 Method II–modify k only

In order to achieve the same search success rate of 90% for the four values of P_b given in Column 1 of Table 3, $k_0 = 32$ must be reduced to a new value k_1 by:

$$k_1 = k_0 \frac{\ln(1 - P_0)}{\ln(1 - P_b)} \tag{16}$$

These values are given in the second column of Table 4. However this means that the new average size of the bucket b_1 increases over the original b_0 by:

$$b_1 = \frac{n}{2^{k_0 - k_1}} = 2^{k_0 - k_1} b_0 \tag{17}$$

Values for hash table accesses and average bucket lengths are given in Table 4. If $|Q_0|$ is the size of the results list resulting from Method II with k_0, and $|Q_1|$ is the size of the results list resulting from Method II with k_1, the relative increase in the size of the results list can be derived from Equations 2 and 17 giving:

$$\frac{|Q_1|}{|Q_0|} = 2^{k_0 - k_1} \tag{18}$$

As we reduce k, the result set increases exponentially.

3.3 Method III–modify L only

The same performance for the four values of P_{BH} given in Column 1 of Table 3 can be achieved by increasing $L_0 = 256$ to L_1 using:

$$L_1 = L_0 \frac{\ln(1 - (1 - P_0)^{k_0})}{\ln(1 - (1 - P_b)^{k_0})} \tag{19}$$

These values are detailed in the second column of Table 5. It can be seen that increasing L to match Method I performance rapidly leads to parameters which are even more impractical to implement than Method I (though the hash table accesses are less than those of extended fingerprint hashing for a range search $r = 3$).

If $|Q_0|$ is the size of the results list resulting from Method III with L_0, and $|Q_1|$ is the size of the results list resulting from Method III with L_1, the relative increase in the size of the results list (by Equation 13) is:

$$\frac{|Q_1|}{|Q_0|} = \frac{L_1}{L_0} \tag{20}$$

Unlike Methods I and II, this method increases the size of the index storage (which is $O(nL)$) as well.

3.4 Method IV–modify k and L

Another option is to change both k and L. We start with a decrease in k_0 by one bit ($k_1 = 31$) and calculate the L_1

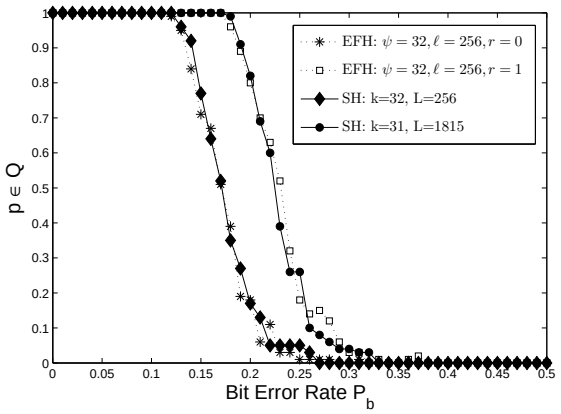

Figure 7: Comparing search success of an extended fingerprint hash to a sample hash on synthetic database of 25,000 uniformly distributed points. Results are averaged over 25 queries.

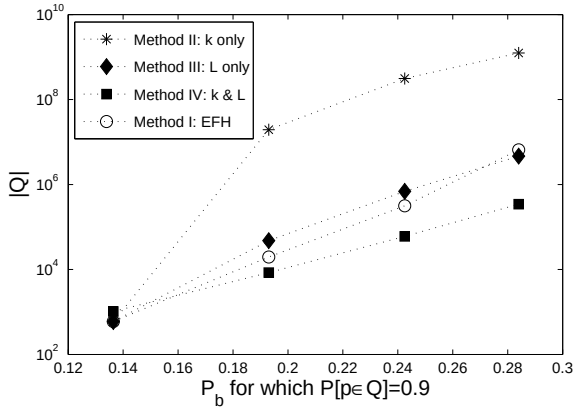

Figure 8: The theoretical size of the results list $|Q|$ retrieved by altering parameters k, L and r while holding $P[p \in Q]$ constant.

needed to achieve the same performance for the four values of P_b given in Column 1 of Table 3 using:

$$L_1 = L_0 \frac{\ln(1-(1-P_0)^{k_0})}{\ln(1-(1-P_b)^{k_1})} \tag{21}$$

which gives us the values for L_1 for Method IV in the second column of Table 6. Table 6 also gives values for the corresponding increases in hash table accesses τ and average bucket size b. If $|Q_0|$ is the size of the candidate results list resulting from Method IV with L_0 and k_0, and $|Q_1|$ is the size of the candidate results list resulting from Method IV with L_1 and k_1, the relative increase in the size of the candidate results list is:

$$\frac{|Q_1|}{|Q_0|} = 2^{k_0-k_1}\frac{L_1}{L_0} \tag{22}$$

With an increase in the storage size of the index proportional to L_1.

We tested the prediction that extended fingerprint hashing and sample hashing would perform similarly using parameters taken from Tables 3 and 6. An extended fingerprint hash and a sample hash with $\psi = k = 32$, $\ell = L = 256$ and $r = 0$ were tested on a database of 25,000 synthetic, uniformly distributed points.

An extended fingerprint hash of $\psi = 32$, $\ell = 256$ and $r = 1$ was tested along with the equivalent sample hash using parameters calculated by Equation 21 to be $k = 31$ and $L = 1815$. Results were averaged over 25 queries. As can be seen from Figure 7, the performance of the extended fingerprint hash and a sample hash are nearly equivalent.

3.5 Summary

The theoretical values of $|Q|$ for each of the four methods in this section are plotted in Figure 8. Methods II and III have a significantly higher search cost than extended fingerprint hash, and we conclude that extended fingerprint hash is superior to these two methods. Method IV has a lower search cost; Method IV returns 43% of an extended fingerprint hash search at $r = 1$ and 19% of an extended fingerprint hash search at $r = 2$ for the same probability of search success.

The reduced search cost of Method IV comes at the cost of an increased index size. A Method IV (for $k = 31$) comparable to an extended fingerprint hash of $r = 1$ has an index seven times larger, and a Method IV ($k = 31$) comparable to an extended fingerprint hash of $r = 2$ has an index 50 times larger.

We looked at various combinations of k and L, but none provides a reduced search cost without a increase in hash-table index size of at least a factor of 5.7. For most applications, this is an unacceptable increase, and we conclude that extended fingerprint hashing is the best option.

There are performance factors which have not been considered here, we have not discussed I/O costs or the dynamic capabilities of these indexes. We have assumed a uniformly distributed set of subfingerprints, and assumed a bit error rate independent of bit position, neither of which may hold for real data. Future work is planned to address these issues.

4. CONCLUSIONS

In this paper we have looked at a fingerprint hash method, initially proposed by Haitsma and Kalker [9], and a sample hash method, initially proposed by Indyk and Motwani [10]. We show how they are related, and develop a new analytic model for predicting search success rate as a function of the bit error rate of the queries. This model was verified by simulation.

For both fingerprint hashing and sample hashing, performance drops sharply as the bit error rate of the queries increases. We look at various ways to improve performance for larger databases and show that changing the size of the sample and the number of samples improves performance for sample hashing, though the index size is increased. Alternatively, one can improve performance by using extended fingerprint hash, a method whereby one searches on all possible bit errors within a Hamming distance of the query.

We compared performance of the two approaches based on the number of candidate fingerprints returned per query and the size of the index. Our analysis shows that if the database index size must be fixed, the number of samples cannot be changed, and extended fingerprint hashing is the superior method. When the number of samples is changed, our anal-

ysis found that a sample hash needed nearly a six-fold increase in database size to produce fewer candidate matches than an extended fingerprint hash for the same search success rate. This may be impractical for many applications, and our conclusion is that the extended fingerprint hash offers the best solution for extending the performance of a hash-based index to queries with higher bit error rates.

While we have analyzed a system specifically within the context of audio fingerprints, many applications in vision, information retrieval and artificial intelligence are beginning to require a high-dimensional database of features or data items. Converting and storing these features as binary strings (as fingerprints) would lead to a more compact database, and comparing binary strings is, in theory, efficient if implemented with bitwise operators. With the analysis given in this paper, the design of a binary hash-based index can be impelemented in a principled way, based on the underlying bit error rate of typical queries presented to the system.

5. ACKNOWLEDGMENTS

The authors wish to thank the Daphne Jackson Trust and the Royal Academy of Engineering for supporting this research.

6. REFERENCES

[1] Audible magic. (Online). Available: [http://www.audiblemagic.com].

[2] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122, 2008.

[3] J. L. Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980.

[4] A. Califano and I. Rigoutsos. Flash: a fast look-up algorithm for string homology. In *Proc. CVPR*, pages 353 –359, jun. 1993.

[5] P. Cano, E. Batlle, T. Kalker, and J. Haitsma. A review of audio fingerprinting. *The Journal of VLSI Signal Processing*, 41(3):271–284, 2005.

[6] P. Cano, E. Batlle, H. Mayer, and H. Neuschmied. Robust sound modeling for song detection in broadcast audio. In *Proc. 112th Int. Conv. of the AES*, 2002.

[7] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[8] K. L. Clarkson. Nearest-neighbor searching and metric space dimensions. In G. Shakhnarovich, T. Darrell, and P. Indyk, editors, *Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*, pages 15–59. MIT Press, 2006.

[9] J. Haitsma and T. Kalker. A highly robust audio fingerprinting system. In *Proc. of the Int. Symposium on Music Information Retrieval*, pages 107–115, 2002.

[10] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. of the 30th Annual ACM Symposium on Theory of Computing*, pages 604–613, New York, NY, USA, 1998. ACM.

[11] R. Karp, O. Waarts, and G. Zweig. The bit vector intersection problem. In *Proc. of the 36th Annual Symposium on the Foundations of Computer Science*, pages 621–630, oct. 1995.

[12] J. M. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *Proc. of the 29th Annual ACM Symposium on Theory of Computing*, pages 599–608, New York, NY, USA, 1997. ACM.

[13] F. Kurth, A. Ribbrock, and M. Clausen. Identification of highly distorted audio material for querying large scale data bases. In *Proc. 112th Audio Engineering Society Convention*, 2002.

[14] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *Proc. of the 30th Annual ACM Symposium on Theory of Computing*, pages 614–623, New York, NY, USA, 1998. ACM.

[15] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*. Cambridge University Press, Cambridge, 2008.

[16] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISSAPP '09: Proc. of the Int. Conf. on Computer Vision Theory and Application*, pages 331–340. INSTICC Press, 2009.

[17] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA '93: Proceedings of the 4th annual ACM-SIAM Symposium on Discrete algorithms*, pages 311–321, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.

[18] P. N. Yianilos. Locally lifting the curse of dimensionality for nearest neighbor search (extended abstract). In *SODA '00: Proc. of the 11th annual ACM-SIAM symposium on Discrete algorithms*, pages 361–370, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[19] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2):6, 2006.