Published in the IEEE Multimedia Signal Processing Workshop 2002, St. Thomas, US Virgin Islands

Audio Fingerprinting: Nearest Neighbor Search in High Dimensional Binary Spaces

Matthew L. Miller
NEC Research Institute
4 Independence Way
Princeton, NJ 08540
Email: mlm@research.nj.nec.com

Manuel Acevedo Rodriguez
EPFL
1015 Lausanne, Switzerland
and
Eurecom Institute
BP 193-06904, Sophia-Antipolis, France
Email: manuel.acevedorodriguez@epfl.ch

Ingemar J. Cox
NEC Research Institute
4 Independence Way
Princeton, NJ 08540
Email: ingemar@ieee.org

***Abstract*— Audio fingerprinting is an emerging research field in which a song must be recognized by matching an extracted "fingerprint" to a database of known fingerprints. Audio fingerprinting must solve the two key problems of representation and search. In this paper, we are given a 8192-bit binary representation of each five second interval of a song and therefore focus our attention on the problem of high-dimensional nearest neighbor search. High dimensional nearest neighbor search is known to suffer from the curse of dimensionality, i.e. as the dimension increases, the computational or memory costs increase exponentially. However, recently, there has been significant work on efficient, approximate, search algorithms. We build on this work and describe preliminary results of a probabilistic search algorithm. We describe the data structures and search algorithm used and then present experimental results for a database of 1,000 songs containing 12,217,111 fingerprints.**

I. Introduction

Audio fingerprinting seeks to recognize a song by extracting a compact representation from an arbitrary, say 5 second, interval and comparing this fingerprint to a database of known fingerprints. There are two primary applications of such technology; usage monitoring to determine broadcast usage fees and media linking in which, for example, a consumer is able to transmit a fingerprint using his or her cell phone to a database and receive back identification information as well as a web site where the song can be purchased.

Audio fingerprinting falls within the domain of classical pattern recognition and must therefore solve both the representation and matching problems. The problem is more tractable than many pattern recognition problems, e.g. three dimensional face recognition, in that the range of distortions is relatively small. Nevertheless, such systems should be scalable to at least one million songs or about 10 billion fingerprints, assuming approximately 10,000 unique fingerprints per song.

Ideally, the fingerprint should be a compact representation that is invariant to a variety common distortions, including additive noise, low pass filtering, subsampling, lossy compression and small offsets in the origin of the fingerprint. Practically, distortions to a song will introduce distortions in the corresponding fingerprint. Obviously, the less the fingerprint is distorted, the easier the subsequent search will be.

In this paper, we assume a representation developed by Philips [1] in which each 5 second interval is represented by an 8192-dimensional binary vector. This vector is the concactenation of 256 32-bit subvectors. Each subvector is derived by taking the Fourier transform of 5/256 second none-overlapping intervals, thresholding these values and subsequently computing a 32-bit hash value. Although each subvector is non-overlapping, the concatenated fingerprints have substantial overlap. Assuming that the duration of the average song is 3 minutes, then the number of fingerprints per song is $3 \times 60 \times 256/5 = 9216 \approx 10,000$.

Given this 8192-dimensional representation, the focus of our work has been to develop an efficient search algorithm. This search can be characterized as a nearest neighbor search in a very high dimensional space. Of course, this assumes that the nearest fingerprint in the database to the query is the correct match. However, under some distortions, this assumption may not be valid. Interestingly, it is less important to correctly match the query to its corresponding fingerprint as it is to match the fingerprint to its corresponding song. Since a song is composed of many fingerprints, incorrectly matching a query to a fingerprint does not necessarily lead to a song recognition error. This issue is discussed in more detail in the experimental evaluation of Section III.

High dimensional nearest neighbor search is a very well studied problem. Proposed solutions generally create a tree structure, the leaf nodes representing the known datum (fingerprints) and searching becomes a traversal of the tree. Specific algorithms differ in how this tree is constructed and traversed. Two related data structures, kd-trees and vantage point or vp-trees, have been extensively studied. However, both data structures succumb to the curse of dimensionality, that is, as the dimension of the datum increases, an increasing percentage of the tree must be searched in order to locate the nearest neighbor to a query.

Recent work [2], [3], [4], [5] appears to acknowledge the fact that a perfect search that guarantees to find the nearest neighbor in a high dimensional space is not feasible. However, the curse of dimensionality can be removed if the search is approximate. For example, Yianilos [5] describes an algorithm that, with probability, p, will find a neighbor within a Euclidean distance r of the query when the datum are uniformly distributed within an n-dimensional hypercube. Unfortunately, this work has not been extended to the binary case and Hamming rather than Euclidean distance.

In this paper, we develop an approximate search algorithm for high dimensional binary vectors. Section II first describes the algorithm. Section III then presents experimental results on a database of 1000 songs and 12,217,111 fingerprints. Finally, Section IV summarizes our results and discusses possible avenues of future work.

II. Algorithm

In the following subsections, we describe an approximate search algorithm for binary vectors in a high dimension space.

Given the set of known fingerprints, we first construct a 256-ary tree. Each 8192-bit fingerprint is represented as 1024 8-bit bytes. The value of each consecutive byte in the fingerprint determines which of the 256 possible children to descend. A path from the root node to a leaf defines a fingerprint.

As the depth of the tree increases, in is common to find nodes with only a single child. This is because the number of actual fingerprints is very much less than the total possible number. For efficiency purposes, we compress suchsequences of nodes with only one child into a single node that represents multiple bytes.

For each level, l, of the tree we then associate a table, T, that will be used to guide our approximate search algorithm. This table is precomputed based on the following theory.

At any node $n_{l,i}$, at level l in the tree, we will have examined b bits of the vector and seen e errors between the first b-bits of the query and the first b-bits represented by the path to node $n_{l,i}$. Then, the probability, p, of observing e errors in b-bits with a bit error rate (BER) of r is a binomial distribution, i.e.

$$p(e|b,r) = \binom{b}{e} e^r b^{(1-r)} \tag{1}$$

assuming that the bit errors are uniformly distributed over the entire fingerprint.

The probability that we will observe *at least* E errors in b-bits is simple one minus the cumulative probability of $p(e|b,r)$, i.e.

$$p(e_o \leq E|b,r) = 1 - \int_0^E p(e|b,r)de. \tag{2}$$

Having observed e errors in b-bits, we can use Equation 2 to calculate the bit error rate r_t, between the query and the closest fingerprint under the node, such that the probability of observing at least e errors is greater than a threshold p_t. That is, r_t is such that $p(e_o \leq E = e|b, r_t) = p_t$. This means that if we have observed e errors in b-bits, we are reasonably certain that the eventual overall bit error rate will be greater than r_t. Thus, if we have already visited a leaf with an error rate below r_t, then we need not search children of this node.

For a given probability threshold, p_t, we can construct a table for r_t indexed by b and e. Such a table is given in Table I for $p_t = 0.05$ and a range of b and e.

0	506	1377	2302	3255	4231	5000	5000	5000
0	257	696	1160	1634	2116	2603	3095	3591
0	172	466	775	1091	1412	1735	2061	2389
0	129	350	582	819	1059	1301	1545	1791
0	103	280	466	656	848	1041	1236	1432
0	86	233	388	546	706	868	1030	1193
0	74	200	333	468	606	744	883	1023
0	65	175	291	410	530	651	772	895

TABLE I

TABLE OF r_t VALUES. THE ROWS ARE THE NUMBER OF BYTES INSPECTED THE COLUMNS ARE THE NUMBER OF ERRORS E. FIRST COLUMN IS FOR 0 ERRORS ENCOUNTERED

Given the tree and associated table, we now describe the search algorithm.

A. Search algorithm

Prior to finding an initial candidate fingerprint for our query, we assume an expected maximum tolerable error rate and assign this to a variable, `best_err_rate`, which represents the best error rate seen so far. For our experiments, this value was 4500.

The search begins by comparing the first byte of the query with the children of the root node. For each child node we calculate the cumulative number of bit errors seen so far. This is simply the sum of the parent errors and the Hamming distance between the 8-bit value represented by the child and the corresponding 8-bit value in the query. Then, in order of increasing error, a test is applied to each child to determine whether to search the child. If the `best_error_rate` is greater than the r_t threshold determined by our table, then the child is searched. This is because the table indicates that we expect the bit error rate under the child to be less than the best rate seen so far. Conversely, if the corresponding r_t threshold is greater than the best error rate seen so far, then we will not search this or subsequent children of the parent node.

The search continues recursively and when a leaf node is reached, the error rate associated with the retrieved fingerprint is compared to the best error rate seen so far. If it is less, then we update the best error rate to this new value and assign this fingerprint as our best candidate nearest neighbor so far.

Pseudo-code for this algorithm is given below.

```
query = signature obtained from input
best_fingerprint = none found
best_err_rate = max err rate tolerable
root.errs = 0
search(root)
search(node)
{
  if node is leaf
  {
    err_rate = node.errs /
               (num bits in query)
    if err_rate < best_err_rate
    {
      best_fingerprint = fingerprint
                         for this node
      best_err_rate = err_rate
    }
  }
  else
  {
    for each child of node
      child.errs = node.errs +
                   hamming distance
                   between child.bits and
                   corresponding bits of query
    for each child of node, in increasing
     order of child.errs
      if best_err_rate >
        thresh_table[child.level][child.errs]
          search(child)
      else
        return
  }
}
```

Note that by defining the table appropriately, we can perform actually perform an exact search that is guaranteed to find the nearest neighbor. Specifically, if for any level, l, of the tree, we set $r_t = e/8192$, then we are assuming that we expect no further error in subsequent levels of the tree. Thus, when a candidate fingerprint is retrieved, all nodes that have fewer errors than the best candidate so far will still be searched.

III. Experimental results

Prior to investigating our search algorithm, we first tested whether fingerprints from one song exhibited any correlation with fingerprints from other songs. Figure 1 illustrates a typical histogram of bit error rates. and clearly shows that inter-song correlation is random, the average bit error rate being 0.5.

We next examined the correlation between fingerprints from the same song. Figure 2 shows that there is a strong temporal correlation in the vicinity of the fingerprint. The duration of this correlation is approximately 0.33 seconds. Fortunately however, even if we match a fingerprint to one of its temporal neighbors, we will still corrcetly identify the song. In fact, the more intra-song correlation that exists, the easier the idenitification becomes.

In order to evaluate our search algorithm, we digitized 1,000 songs and computed 12,217,111 corresponding fingerprints.

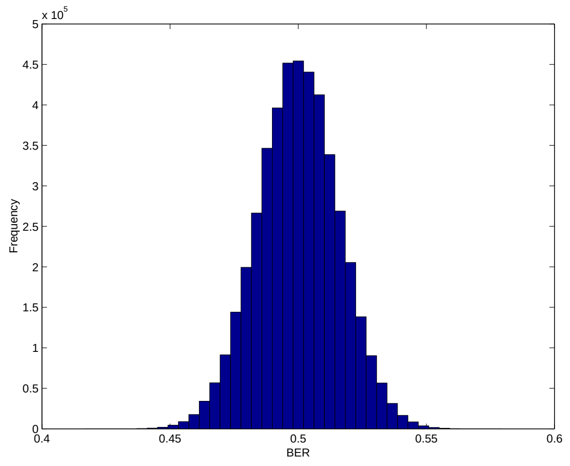

Fig. 1. Histogram of bit error rates between a fingerprint from one song and the fingerprints of all other songs.

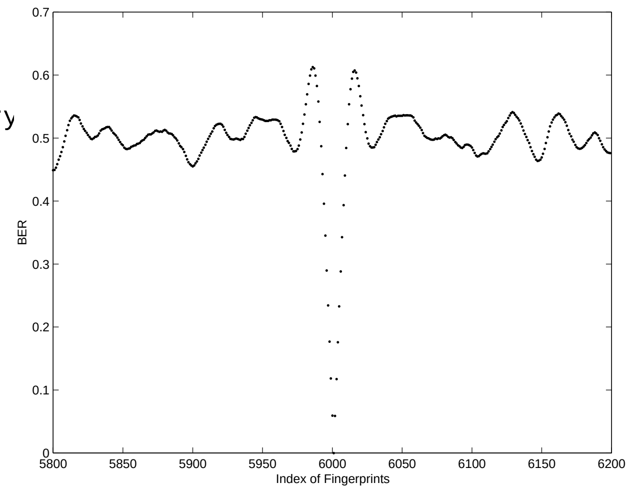

Fig. 2. Computed Hamming distance between a fingerprint from one song and all the fingerprints from the same song.

4913 queries were generated by randomly selecting 5 second portions from the song database and playing them through inexpensive speakers attached to a PC. These song snippets were then digitized using an inexpensive microphone.

For each query, an exhaustive search of the database was performed in order to determine the closest fingerprint in the database. This provided ground truth data with which to compare with our approximate search results. The distribution of distances is shown in Figure 3. We see that the Hamming distance between the majority of queries and their nearest neighbors is approximately 1065, which is equivalent to a bit error rate of 13%. However, the range of Hamming distances is between 423 and 3532 with corresponding bit error rates of 5% and 43% respectively.

Since we know which song each query is derived from, we were also able to determine the validity of our assumption that the nearest neighbor identifies the correct song. In 53 cases, the nearest neighbor did not correctly identify the song. This

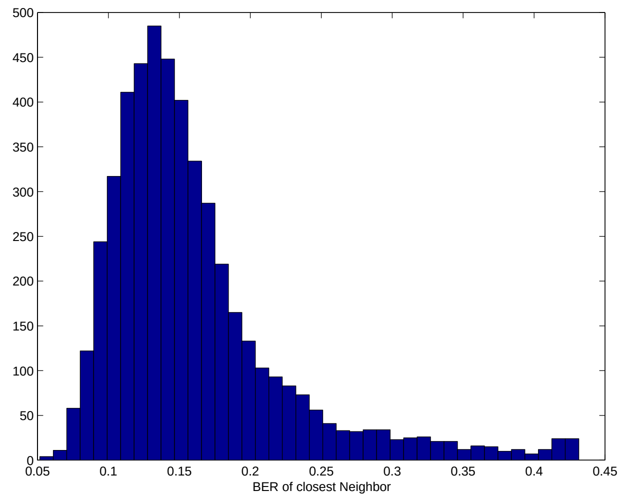

Fig. 3. Distribution of distances between queries and their closest fingerprints in the database.

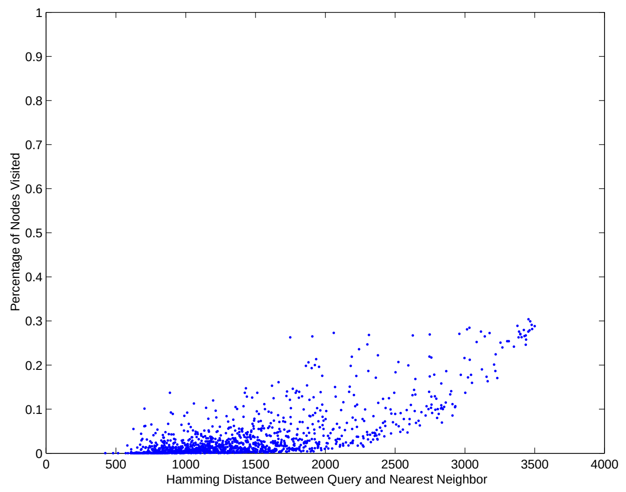

Fig. 4. Number of nodes examined as a function of the known distance between a query and its nearest neighbor in the database.

is approximately a 1% error rate.

A. *Search results*

Different thresholds for p_t and corersponding tables for r_t will produce different search statistics. Typically, the smaller the threshold p_t, the closer our approximation will be to an exact search, but this comes at the expense of searching more nodes in the tree. Clearly, we seek to find a compromise threshold in which acceptable retrieval accuracy is obtained while searching as little of the tree as possible.

Here, we describes results in which $p_t = 0.40$. With the queries drawn from the distribution of Figure 3, we search an average of 211,620 nodes, which is 4.85% of the nodes in the tree. On average, only 55% of the correct nearest neighbors are found. However, the erroneous song rate is only 3%. We attribute this discrepancy to the significant temporal correlation between fingerprints in a song.

Clearly, the distribution of queries can have a very significant effect on the overall search performance. To investigate this effect, we tabulated the number of nodes examined as a function of the distance between the query and its nearest neighbor. Figure 4 shows that for queries with nearest neighbors less than a Hamming distance of 2000, the percentage of nodes visited is much less than 10%. However, as the Hamming distance increases from 2000 to 3500 a much greater fraction of the database must be searched.

If only queries with a Hamming distance of 2900 or less are considered, then the erroneous song detection drops to 1% and the average number of nodes visited is 157,960 or 3.62%. The number of correctly identified fingerprints does not improve significantly and is 56%. However, as previously noted, it is the song detection rate that is important.

IV. Discussion

We described preliminary results for an approximate search algorithm which can be used to identify songs based on 5 second samples. These samples are represented as a 8192-bit vector and we proceeded to develop an approximate search algorithm for high-dimensional nearest neighbor search.

This search algorithm constructs a 256-ary tree and an associated table that is constructed assuming that the errors between a query and its nearest neighbor are uniformly distributed. Branches of the tree are pruned by comparing the bit error rate of the current best candidate with the likely bit error rate we expect from the nearest neighbor below a node in the tree.

For a database of 1000 songs and 12,217,111 fingerprints, we demonstrated a song recognition rate of 97% while on average only searching 4.85% of the nodes in the tree. Queries can be more or less noisy and the distribution of queries can significantly affect the search. If only queries with a Hamming distance of less than 2867 to their nearest neighbor are considered, then we are able to achieve a song recognition rate of 99% while only visiting and average of 3.62% of the tree.

Significant future work is possible. This includes studying the effect of the threshold probability, p_t on the song recognition rate and investigating how the performace scales with the size of the song database. We also believe that a more accurate bayesian model of the search process might be formulated that would provide significant performance increases.

References

[1] T. Kalker, "Personal communication," 2002.
[2] J. M. Kleinberg, "Two algorithms for nearest-neighbor search in high dimensions," in *Proc. of the 29th annual ACM Symp. on Theory of Computing*, 1997, pp. 599–608.
[3] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. of the 30th annual ACM Symp. on Theory of Computing*, 1998, pp. 604–613.
[4] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching in fixed dimensions," *J. of the ACM*, vol. 45, no. 6, pp. 891–923, 1998.
[5] P. N. Yianilos, "Locally lifting the curse of dimensionality for nearest neighbor search," in *Proc. of the 11th annual ACM-SIAM Symp. on Discrete Algorithms*, 2000, pp. 361–370.