A Tree-Based Method For Fast Melodic Retrieval

Charles Parker
Oregon State University
Department of Electrical Engineering and Computer Science
Dearborn Hall Rm. 102, Corvallis, OR, 97330
parker@cs.orst.edu

ABSTRACT

The evolution of aurally queryable melodic databases (so-called *query-by-humming* systems) has reached a point where retrieval accuracy is relatively high, even at large database sizes. With this accuracy has come a decrease in retrieval speed as methods have become more sophisticated and computationally expensive. In this paper, we turn our attention to heuristically culling songs from our database that are unlikely given a sung query, in hopes that we can increase speed by reducing the number of matching computations necessary to reach the proper target song.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Retrieval

General Terms

Algorithms, Performance

Keywords

Melodic retrieval, query-by-humming

1. INTRODUCTION

Over the past 10 years, a body of literature has accumulated around the idea of an aurally queryable music database [2]. More precisely, we wish to develop a database which contains a listing of songs along with the musical notes that compose each song. We then imagine that a user can sing into a computer which, by some combination of signal processing and a similarity function, retrieves the proper song from the database based on the sung query.

Unfortunately, several issues within the problem make it a non-trivial one. First, we must extract notes from a recording of a person singing. This is difficult enough, but we are also faced with the fact that singers will almost certainly not sing the song perfectly. By the time we reach the matching phase of the process we are given a string of notes that may be riddled with insertion, deletion, and substitution errors.

The attempts to get around this problem are many and varied. Most are centered on generating a probabilistic score between each target song in the database and the sung query, and then using this to rank the targets. An approach [5] that is currently *en vogue* is an HMM-based model. Although it performs admirably, Dannenburg [1] reports that matching a single target song against a sung query takes around two seconds. With even a modestly sized database of 10,000 songs, this process would take over five and a half hours.

Intuitively, however, this "linear" form of matching seems unnecessary. If we have a system of probabilistically matching one song to another, we ought to be able not only to match targets to queries, but also targets to targets, the idea being a sort of transitivity in our matching scheme: If a sung query is nothing like a given target t_1, it is within reason to assume that another target, t_2, that is very similar to t_1 will likely not be a good match either.

Some work has been done along these lines using *suffix trees* [6], but such an attempt is limited to those systems with string matching as their core algorithm, ignoring HMM-based [5] and frame-based [4] alternatives that perform as well or better [1]. The next section attempts a more general approach.

2. TREE-BASED MATCHING

We define a function $P(x, y)$ which, given two sequences of notes x and y, returns a number in the range $(0, 1)$ which increases as x and y are more similar. Let us suppose that we have a set of such sequences, or songs, and call this set $\mathcal{S}$. We can divide this set into two subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ by finding the two least similar songs in $\mathcal{S}$, d_1 and d_2. That is,

$$\exists_{d_1,d_2\in\mathcal{S}}\forall_{x,y\in\mathcal{S}}P(d_1, d_2) \leq P(x, y)$$

Then, we can partition $\mathcal{S}$ into $\mathcal{S}_1$ and $\mathcal{S}_2$ as follows:

$$\forall_{x\in\mathcal{S}}P(x, d_1) > P(x, d_2) \Rightarrow x \in \mathcal{S}_1$$

So that for a given song in $\mathcal{S}$, it belongs to $\mathcal{S}_1$ if is more similar to d_1 than it is to d_2, and similarly for $\mathcal{S}_2$. Obviously, we can repeat this procedure recursively to form a tree. If we limit the depth of this tree, the leaf nodes contain sets of songs rather than individual songs.

Of course, this is not bound to work. It is based on the intuition that a query will follow the same path down the tree as a target if the two are similar enough, and that songs in general have enough of a difference between them to support such a discrimination. We would, however, like to introduce a "forgiveness" factor so that weakly similar songs are not lead down the wrong subtree. We do this both by limiting

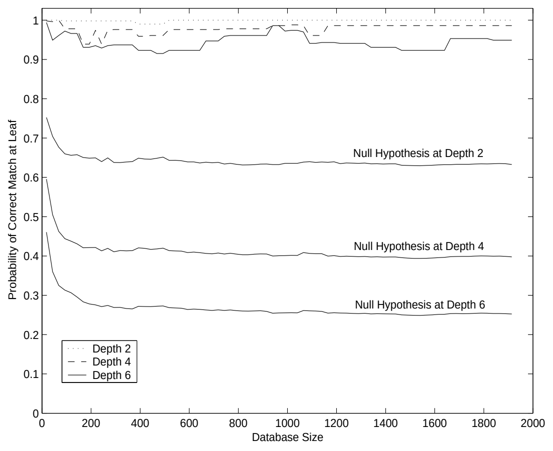

Figure 1: Success rate vs. database size at varying tree depths.

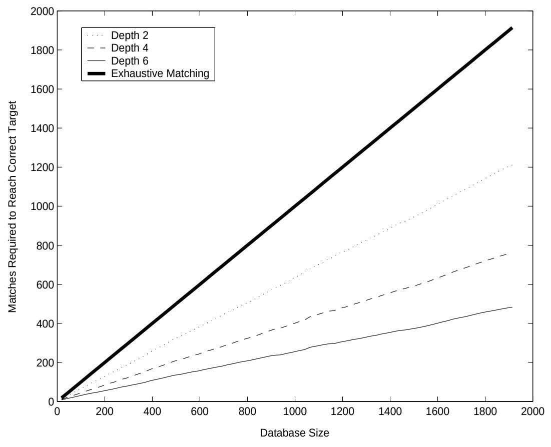

Figure 2: Leaf size vs. database size at varying tree depths.

the depth of the tree and by letting songs in $\mathcal{S}$ that are not strongly biased to either d_1 or d_2 become members of both $\mathcal{S}_1$ and $\mathcal{S}_2$. We define "strong bias" to mean a better than average distance between $P(x, d_1)$ and $P(x, d_2)$. Formally, we allow x membership in both $\mathcal{S}_1$ and $\mathcal{S}_2$ if and only if:

$$|P(x, d_1) - P(x, d_2)| < \frac{\sum_i |P(d_1, s_i) - P(d_2, s_i)|}{|\mathcal{S}|}$$

where s_i is the ith song in $\mathcal{S}$. Also note that the vertical bars denote cardinality for $\mathcal{S}$ and absolute value otherwise.

We are still at a loss to actually prove, however, that a given query will follow the correct target to the proper subtree. This we deal with empirically.

3. EXPERIMENTAL SETUP

To test the effectiveness of this method, we gathered about 550 queries over 12 different songs and 50 different singers, most of whom could be described as "hobbyists" - singers without classical training. Of these, 507 queries were deemed reasonably accurate, which we defined to mean among the top three matches on a small database of 18 songs. We then constructed trees of various depths and database sizes according to the method above

For our function $P(x, y)$, we abandon the time consuming HMM-based approach in favor of the simpler one presented by Hu [3], with the exception that ours is note-based rather than frame-based. Signal processing and note segmentation was done largely with techniques outlined in [5].

4. RESULTS AND CONCLUSIONS

We plot the results in the two figures above. Figure 1 shows the probability of a query reaching the correct leaf in the tree at several depths. For perspective, we have also plotted the *null hypothesis*, the expected probability of reaching the correct leaf had we chosen to simply eliminate random targets from the database at each depth level.

We see that, even at depth level six, we still lead better than 90% of all of our queries to the proper leaf in the tree. Figure 2 shows that we are able to effectively ignore about 75% of the database at this depth level - that is, the size of the sets at the leaves at depth six is about a quarter of the whole database. We again plot, for perspective, the number of matching computations we would perform without the tree-based solution.

But the real promise of the technique lies in its vast possibilities for modification: We have an obvious trade-off between the likelihood of being lead to the proper leaf node and the number of songs at each leaf. There may be more effective schemes for forgiveness, and these can be "plugged-in" to the algorithm. In addition, we may similarly plug-in any function $P(x, y)$ to build the tree, any number of which may yield better results than those shown here.

In summary, we have shown a basic tree-based method for melodic retrieval, and that it can effectively cut down the number of matching computations required to locate the correct target song without a substantial reduction in retrieval accuracy. In addition, we believe that the extensibility of the method makes it a promising candidate for future investigation.

5. REFERENCES

[1] R. B. Dannenberg, W. P. Birmingham, G. Tzanetakis, C. Meek, N. Hu, and B. Pardo. The musart testbed for query-by-humming evaluation. In *Proc. 4th International Symposium on Music Information Retrieval*, 2003.

[2] A. Ghias, J. Logan, D. Chamberlin, and B. C. Smith. Query by humming: Music information retrieval in an audio database. In *Proc. 3rd ACM Multimedia Conference*, pages 231–236, 1995.

[3] N. Hu, R. Dannenburg, and A. L. Lewis. A probabilistic model of melodic similarity. In *Proc. International Computer Music Conference*, 2002.

[4] D. Mazzoni and R. B. Dannenberg. Melody matching directly from audio. In *Proc. 2nd Annual International Symposium on Music Information Retrieval*, 2001.

[5] C. Meek and W. Birmingham. Johnny can't sing: A comprehensive error model for sung music queries. In *Proc. 3rd International Symposium on Music Information Retrieval*, 2002.

[6] S. Pauws. Cubyhum: A fully operational query by humming system. In *Proc. 3rd International Symposium on Music Information Retrieval*, 2002.