# A SOM-based Multimodal System for Musical Query-by-Content

Kyle Dickerson and Dan Ventura

*Abstract*— **The ever-increasing density of computer storage devices has allowed the average user to store enormous quantities of multimedia content, and a large amount of this content is usually music. We present a query-by-content system which searches the actual audio content of the music and supports querying in several styles using a Self-Organizing Map as its basis. Empirical results demonstrate the viability of this approach for musical query-by-content.**

## I. Introduction

Many personal libraries contain thousands of songs which the user needs to search through when looking for a particular song. Current search techniques typically rely on meta-data tags which describe artist, album, year, genre, or similar information. These tags must be created by a human and attached to each file—an error-prone process which is, at best, inconvenient.

Much work has been done to create systems which try to automatically tag a song with genre information [1]. Such meta-data is helpful, but only if the user can remember the information stored in the tags. Otherwise it is necessary to search by content, and such systems, which rely on information retrieved from audio files, are generally referred to as Music Information Retrieval (MIR) systems. Unfortunately, no system yet exists that searches audio by content and which is accurate, fast, robust, and intuitive.

Any MIR system will require a method for determining the similarity of songs. In fact, the system is heavily dependent on this distance function. Many current systems first transcribe the audio content to a text representation and then use common string-matching techniques as the distance function [2], [3], [4]. This process, however, is problematic because it reduces the content-rich music to a simple text string.

Instead, one could in principle extract various acoustic features from the audio using signal processing techniques with the distance function dependent upon which musical features are used. Determining which features to extract is a difficult, context dependent problem [5], [6], [7], [8], [9].

Once a good feature set is found, it is still necessary to determine a suitable distance function. The choice of distance function has also been heavily studied, resulting in varying levels of success [10], [11], [12]. However, instead of explicitly constructing a distance function, another approach would be to implicitly derive one. Self-Organizing Maps (SOMs) produce an interpolated feature space and can easily be visualized, and, as a result, naturally suggest themselves as a way of doing this. In earlier work, we proposed the design of a system that uses a SOM to reduce the dimensionality of the feature space, allowing the use of simple (Euclidean) distance metrics [13]. Here we extend that work by fully implementing the proposed system and evaluating it for a variety of different query scenarios.

Kyle Dickerson and Dan Ventura are with the Department of Computer Science, Brigham Young University, Provo, Utah (email: kyle.dickerson@gmail.com, ventura@cs.byu.edu).

## II. Related Work

Our work combines two well-established concepts—Self-Organizing Maps and Musical Query-by-Content Systems. The Self-Organizing Map is an unsupervised learning algorithm which uses data from a feature set of any dimensionality to generate a two-dimensional grid (map) while attempting to preserve as much of the intrinsic similarity of the data as possible [14]. The learning of such a map (see Algorithm 1) begins by initializing a grid of random feature vectors (line 1). This grid may be considered to wrap around both horizontally and vertically, creating a toroid, to prevent unusual edge effects. For each training datum (line 3), the grid location of the most similar map vector is found (line 4), and all vectors in a neighborhood around the matching vector are updated to become more like that datum (lines 5-8). Over time the size of the neighborhood shrinks (line 9) and the influence of the update decreases (line 10). In effect, these neighborhood alterations create smooth interpolations between data points across the map, a desirable property which allows us to train on a subset of the available data and still get a useful map.

SOMs have already been used successfully in MIR systems. One of the first such systems was presented by Feiten and Günzel [15]. Harford [16] uses a SOM to perform melody retrieval. Dittenbach, Merkl, and Rauber [17] introduce a growing hierarchical SOM which Rauber, Pampalk, and Merkl [18] use to create a musical archive based upon sound similarity. As far as we have been able to determine, no work has been done in applying SOMs to a musical query-by-content system. However, an image query-by-content system was created by Laaksonen, Koskela and Oja [19]. In this system the user does not input a free-form query but rather selects images from presented sets as the system locates the area of the SOM the user is interested in.

The textual representation of the query is a considerable constraint in all of the above systems. Music is a very rich medium—songs often contain concurrent parts and a user may remember any single part while forgetting the others. Having a strong, unique bass with a vocal track accompanied by instrumentals is not uncommon, but by reducing the song to a single text representation much of that information is lost. Perhaps a user can remember how the bass sounded, but not the vocal or instrumentals. Current systems would be unable to help them find the correct song. One system does allow more comprehensive searching by extracting features

**Algorithm 1** *SOM algorithm pseudocode.* $X$ is the training set of real-valued vectors of length $n$. $m$ is the size of the SOM (square $m \times m$). $\rho \in [1, m]$ is the radius used in determining neighborhoods. $\alpha \in (0, 1)$ is the weight given to the training datum when updating the vectors on the grid. $c_\rho, c_\alpha \in (0, 1)$ are linear decay multipliers for $\rho, \alpha$. The vector and grid distance functions can be any metric—here, Euclidean distance is used for both. Typical parameter values: $\alpha = 0.1$, $\rho = \frac{m}{2}$, $c_\rho = 0.999$, $c_\alpha = 0.999$.

```
Train($X, m, \rho, \alpha, c_\rho, c_\alpha$)
1:  $M \leftarrow m \times m$ grid of random real-valued vectors of length $n$
2:  while NOT DONE do
3:    for $\bar{x} \in X$ do
4:      $\bar{v} \leftarrow \text{argmin}_{\bar{v} \in M}$ vector_distance $(\bar{v}, \bar{x})$
5:      $v_i \leftarrow \alpha x_i + (1 - \alpha) v_i$
6:      for $\bar{u}$ in neighborhood($\bar{v}, \rho$) do
7:        $\alpha_u \leftarrow \alpha \times \frac{\rho - \text{grid\_distance}(\bar{v}, \bar{u})}{\rho}$
8:        $u_i \leftarrow \alpha_u x_i + (1 - \alpha_u) u_i$
9:    $\rho \leftarrow c_\rho \rho$
10:   $\alpha \leftarrow c_\alpha \alpha$
11: return($M$)
```

directly from the MP3 encoding format which are segmented into a set of "phases" to which queries are matched [20]. This work, however, is limited to only songs in MP3 format. Cui et al. create a hybrid system combining metadata and query-by-content to produce a single result set [21]. This is a compelling approach; however it could be applied with any underlying systems and does not eliminate the need to develop better standalone query-by-content systems.

We use a SOM to power a musical query-by-content system, thereby allowing us to retain as much of the original audio content as possible. By retaining more content in the target songs, the system is amenable to search using a broader range of query types: humming, whistling, singing, etc.

## III. System Design

The system design incorporates a unique path-descriptor-based approach to music processing that incorporates a SOM built to organize audio signals. Figure 1 illustrates the process. Algorithm 2 gives pseudocode for building a library of song path-descriptors from a library $L$ of songs. First, a SOM is created from randomly sampled song segments (lines 1-3). Before the SOM is trained, the audio is preprocessed (line 2) to extract feature vectors, using features and parameters that are common in many MIR systems [22], [23], including the power spectrum, strongest beat, beat sum, Mel-Frequency Cepstral Coefficients, and Linear Predictive Coding.

Once the SOM is trained, each of the songs $l$ in the library$L$ (line 5) is preprocessed in the same way as the training segments, resulting in an ordered set $X$ of feature vectors that correspond to song subsegments of length $\delta$ (line 6). For each of these feature vectors (line 7), the SOM vector that is most similar is found (line 8) and the SOM

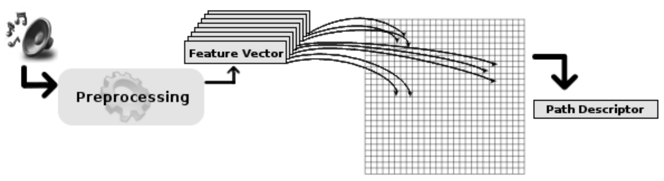


Fig. 1. *High-level system design.* During training, 9 consecutive feature vectors (representing 25 seconds of audio) are taken from 176 (20%) of the preprocessed songs (1584 unique feature vectors). The feature vectors are used as the training data for the SOM algorithm. After training has completed, each song's complete set of feature vectors is mapped into the SOM creating unique path-descriptors.

**Algorithm 2** *Path Library pseudocode.* $L$ is a library of songs. $k \in (0, 1)$ is the fraction of the library to use for training. $\lambda$ is the length of song samples. $\delta$ is the subsampling window size. $\theta$ is the fraction of overlap for subsample windows. Preprocessing can involve any audio signal processing that produces real-values that can be composed as a vector—here features used include the power spectrum, MFCCs and Linear Predictive Coding. For the results reported here, $L$ is a library of 881 songs, $k = 0.2$, $\lambda = 25$ seconds, $\delta = 5$ seconds and $\theta = 0.5$.

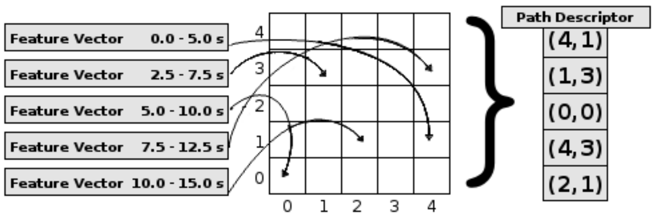
```
Build_Path_Library($L, k, \lambda, \delta, \theta$)
1:  $S \leftarrow kL$ random song segments of length $\lambda$ from $L$
2:  $X \leftarrow$ Preprocess($S, \delta, \theta$)
3:  $M \leftarrow$ Train($X, m, \rho, \alpha, c_\rho, c_\alpha$)
4:  $P \leftarrow \emptyset$
5:  for $l \in L$ do
6:    $X \leftarrow$ Preprocess($l, \delta, \theta$)
7:    for $i = 1$ to $|X|$ do
8:      $a_i \leftarrow \text{argmin}_{(j,k)} ||M_{jk} - x_i||_2$
9:    $a_l \leftarrow (a_1, \ldots, a_{|X|})$
10:   $P \leftarrow P \cup \{a_l\}$
11: return($P$)
```

coordinates of this vector are recorded as a tuple, $a_i$ (line 8). This ordered set of tuples, $a_l = (a_1, \ldots, a_{|X|})$ is the path-descriptor representation of song $l$ (line 9) and is stored in the descriptor library (line 10).

Figure 2 illustrates the path-descriptor concept. As usual for SOMs, each data instance (vector of audio features) will map to a single location within the SOM; but, because each song/sample is represented by an ordered set of feature vectors, the song/sample can be represented as the ordered set of locations to which the individual vectors map. This allows for cleanly handling the temporal nature of the problem without the need to change the actual SOM algorithm in any way. These paths now act as unique descriptors for each song and are stored for later reference (line 11).

The distance between two songs, $l_1$ and $l_2$ is calculated as the average Euclidean distance between the points composing their unique path-descriptors, $p_1$ and $p_2$:

$$d(l_1, l_2) = \frac{1}{m} \sum_{i}^{m} ||p_{1_i} - p_{2_i}||_2 \tag{1}$$



Feature Vector 0.0 - 5.0 s
Feature Vector 2.5 - 7.5 s
Feature Vector 5.0 - 10.0 s
Feature Vector 7.5 - 12.5 s
Feature Vector 10.0 - 15.0 s
4
3
2
1
0
0 1 2 3 4
Path Descriptor
(4,1)
(1,3)
(0,0)
(4,3)
(2,1)

Fig. 2. *Path-descriptor example.* After the SOM has been trained, the feature vectors representing a song can be mapped into the SOM. Each feature vector maps to a single location and the ordered set of locations produces a unique path-descriptor describing that song. These path-descriptors are then used to compute the distance between songs (see Figure 3).

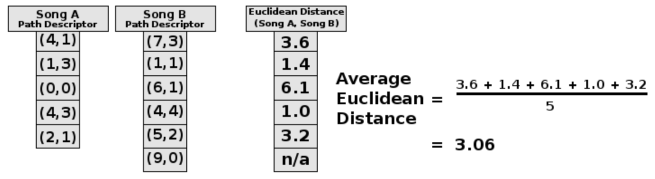


Fig. 3. *Song distance calculation example.* The distance between two songs is calculated as the average Euclidean distance between the points composing their unique path-descriptors. If the path-descriptors have different lengths, the extra points in the longer descriptor are ignored. This very simple technique yields fairly good results.

where $m = \texttt{min}(\texttt{len}(p_1), \texttt{len}(p_2))$. When the descriptors vary in length, the extra points in the longer descriptor are simply ignored (see Figure 3).

## A. Quasi-Supervised SOM Querying

Because we allow the user to query using their choice of modality, the query is not likely to be similar to any song based upon the musical content (other than the one theme expressed in the query). For example, whistled queries are unlikely to match musical content occurring from guitars, singing, pianos, etc. As a result, whistled queries (for example) are likely to sound more like each other (to the naïvely trained SOM) than like their intended target songs. The solution is a quasi-supervised approach to SOM training that allows comparison of songs and queries using a single SOM, while avoiding this problem.

Traditionally SOM training is unsupervised—the feature vectors themselves determine the resulting map. In quasi-supervised training, song segments are paired with matching user queries to produce the vectors used to train the SOM. Algorithm 3 shows how Algorithm 2 is modified to accomplish this. The feature vectors of the song segments are augmented with the feature vectors of matching sample user queries (lines 1-4). The actual process of training the SOM is no different than before—each training vector is considered and the neighborhood around the closest matching location is updated. That is, as far as the actual SOM is concerned, nothing has changed (line 8). What has changed is how the vectors that compose the SOM are viewed. While the SOM is still presented with simple vectors for training, half of the vector is interpreted as a representation of the features extracted from the actual song and the other half is interpreted as a representation of the features extracted from the sample query (line 7). For example, if the feature vectors for the songs each contain 13 values and the feature vectors for the queries each contain 13 values, then the vectors in the SOM will each contain 26 values. This process will create a single interpolated map linking queries and targets. Figure 4 illustrates the high-level idea.

**Algorithm 3** *Quasi-supervised Path Library pseudocode.* $L$ is a library of songs. $R = \{(s_i, q_i)\}$ is a set of song segment/query pairs. $\delta$ is the subsampling window size. $\theta$ is the fraction of overlap for subsample windows. Since the dimensionality of $M_{jk}$ is twice that of $x_i$, the superscript indicates with which half of the vector $M_{jk}$ the $L_2$-norm should be computed.

```
Build_Quasi_Path_Library($L, R, \delta, \theta$)
 1: $S \leftarrow \{s_i | (s_i, q_i) \in R\}$
 2: $Q \leftarrow \{q_i | (s_i, q_i) \in R\}$
 3: $Z \leftarrow$ Preprocess($S, \delta, \theta$)
 4: $Y \leftarrow$ Preprocess($Q, \delta, \theta$)
 5: $X \leftarrow \emptyset$
 6: for $i = 1$ to $|R|$ do
 7:     $X \leftarrow z_i \circ y_i$ {concantenation}
 8: $M \leftarrow$ Train($X, m, \rho, \alpha, c_\rho, c_\alpha$)
 9: $P \leftarrow \emptyset$
10: for $l \in L$ do
11:     $X \leftarrow$ Preprocess($l, \delta, \theta$)
12:     for $i = 1$ to $|X|$ do
13:         $a_i \leftarrow \text{argmin}_{(j,k)} ||M^1_{jk} - x_i||_2$
14:     $a_l \leftarrow (a_1, \ldots, a_{|X|})$
15:     $P \leftarrow P \cup \{a_l\}$
16: return($P$)
```

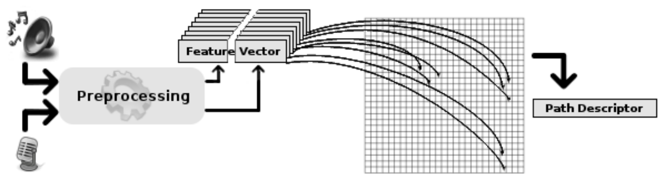


Fig. 4. *Quasi-supervised SOM design.* Sample queries are matched with their target songs. These pairs are individually preprocessed and the resulting pair of feature vectors are concatenated to create a single feature vector, which is used to train the SOM. After training, the songs in the library are mapped into the SOM using the first half of the stored feature vectors to create path-descriptors. Queries are similarly mapped, using the second half of the stored feature vectors. The path-descriptors of targets and queries are thus directly comparable while still explicitly modeling the query style.

Once the SOM is trained, path-descriptors are generated for all the songs in the music library, just as was done in Algorithm 2, with one small change: when the vectors describing a song are compared to the SOM, they are only compared to the first half of the vectors in the SOM (line 13, indicated with superscript 1).

Given a SOM trained using Algorithm 3 and the resulting library $P$ of song path-descriptors, one can now query the song library, taking advantage of the augmented vectors in the SOM. Algorithm 4 shows how this is accomplished.

**Algorithm 4** *Path Library Querying pseudocode.* $r$ *is a user query.* $P$ *is a library of song path-descriptors. Since the dimensionality of* $M_{jk}$ *is twice that of* $x_i$*, the superscript indicates with which half of the vector* $M_{jk}$ *the* $L_2$*-norm should be computed.*

```
Query_Path_Library($r, P$)
 1: $X \leftarrow$ Preprocess($r, \delta, \theta$)
 2: for $i = 1$ to $|X|$ do
 3:     $q_i \leftarrow \text{argmin}_{(j,k)} ||M^2_{jk} - x_i||_2$
 4: $q \leftarrow (q_1, \dots, q_{|X|})$
 5: $closest \leftarrow \infty$
 6: for $p \in P$ do
 7:     $dist \leftarrow \text{argmin}_{w \in p}\, d(w, q)$ {$w$ is window the length
        of $q$}
 8:     if $dist < closest$ then
 9:         $answer \leftarrow p$
10: return($answer$)
```

First, a path-descriptor for the query is generated (lines 1-4) by comparing the vectors describing the query to only the second half of the vectors in the SOM (indicated with a superscript 2 in line 3). Once the path-descriptor for the query is generated, it can be directly compared to the previously computed path-descriptors for all the songs in the library (lines 6-9). The distance between path-descriptors in line 7 is computed using Equation 1 and represents how similar a query is to a song.

## IV. Empirical Results

The goal of any querying system is to return the specific item a user is searching for as the number one result. Therefore, the quality of the Query-by-Content system can be evaluated using the position of the true target within the ordering of the results—the closer to the first result the better. Given a query $q \epsilon Q$ and an ordered list of results for $q$, $\tau(q)$ returns the index of the correct target song for $q$ in the ordered results for $q$. Values for $\tau(q)$ range from 0 (a correct match) to $|T| - 1$, where $T$ is the set of target songs. The percentage of targets occurring as the first result are computed using Equation 2. Two additional metrics are the percentage of targets occuring within the top 5 results (Equation 3) and top 10 results (Equation 4) [2]. These metrics are commonly used because, considering the difficulty of the task, requiring the user to listen to five or ten possible matches is not unreasonable when searching through a database of possibly thousands of songs. The final metric that will be used is the average position of the target in the ordered results (see Equation 5).

$$\%\,Correct = \left( \frac{\sum_{q \epsilon Q} \delta(\tau(q), 0)}{|Q|} \times 100 \right) \% \qquad (2)$$

$$\%\,in\,Top\,5 = \left( \frac{\sum_{q \epsilon Q} \delta(\lfloor \frac{\tau(q)}{5} \rfloor, 0)}{|Q|} \times 100 \right) \% \qquad (3)$$

$$\%\,in\,Top\,10 = \left( \frac{\sum_{q \epsilon Q} \delta(\lfloor \frac{\tau(q)}{10} \rfloor, 0)}{|Q|} \times 100 \right) \% \qquad (4)$$

$$Average\ Position = \frac{\sum_{q \epsilon Q} \tau(q)}{|Q|} \qquad (5)$$

For evaluation, the average values of the four metrics are compared to the baseline of expected values of those metrics if given random orderings. For example, given 15 songs in the training set the expected value of the 0-based position of the target song in a random ordering would be $\frac{15-1}{2} = 7$. The general form of the expected values of the percentage of songs correct, percentage of songs in the top 5, percentage of songs in the top 10, and average position are presented in Equations 6, 7, 8, and 9, respectively.

$$E[\%\,Correct] = \left( \frac{1}{numSongs} \times 100 \right) \% \qquad (6)$$

$$E[\%\,in\,Top\,5] = \left( \frac{5}{numSongs} \times 100 \right) \% \qquad (7)$$

$$E[\%\,in\,Top\,10] = \left( \frac{10}{numSongs} \times 100 \right) \% \qquad (8)$$

$$E[Average\ Position] = \frac{numSongs - 1}{2} \qquad (9)$$

### *A. Data Collection*

Sample queries were collected from eight test-users, and each user chose the query style with which they felt most comfortable (see Table I). To begin the query collection process, a song is randomly chosen from the song library, and then a 10-second segment of that song is randomly selected. The user is asked to listen to the 10-second clip until they feel comfortable that they know the clip. The user then records a query sample of that clip while listening to the clip again. To prevent the true signal from being included in the query samples, the audio is only played through headphones. The query sample is captured using an Apex 181 USB condenser microphone at 44.1 khz. This process was repeated during two 1-hour sessions at a pace set by the user. All of the audio was then processed to extract a desired feature set, yielding a set of eight datasets of matched query-sample pairs $(q, s)$, where $q$ is the features representing ten seconds of humming, and $s$ is the features representing the ten seconds of actual audio sample the user was trying to match. These 8 datasets varied in size from 26 to 210 pairs.

### *B. Experimental Design*

Using these eight datasets, 5-fold cross validation-based testing was performed for a variety of scenarios involving different audio features, single vs. multiple users and single vs. multiple query styles. Each dataset was randomly divided into 5 groups as equally as possible (each group differing by no more than one song). Each fold (approximately 20% of the dataset) was used once for training and appeared in the test

| User | Music Skill | Training | Query Style | Query Skill |
|---|---|---|---|---|
| A | 5 | None | Sing (words) | 4 |
| B | 6 | None | Hum | 6 |
| C | 7 | Some | Sing (words) | 6 |
| D | 4 | Some | Hum | 5 |
| E | 7.5 | Lots | Sing (notes) | 6 |
| F | 3 | Little | Whistle | 5 |
| G | 7 | Some | Whistle | 7 |
| H | 6 | None | Sing (words) | 6 |

TABLE I

*Test-user survey information.* TEST-USERS PROVIDED INFORMATION ABOUT HOW MUCH MUSICAL TRAINING THEY HAVE HAD, THE STYLE OF QUERYING THEY WOULD USE, AND RATED THEMSELVES ON THEIR OVERALL MUSICAL SKILL AND SKILL IN THEIR QUERY STYLE. THESE SKILL RATINGS ARE ON A SCALE OF 1-10 WHERE 10 IS THE BEST.

set of all other folds, yielding test sets of approximately 80% of the dataset. The evaluations were run twice, once with a feature set of only the Mel-Frequency Cepstral Coefficients (MFCCs), and once with a feature set of the MFCCs and the Linear Predictive Coding (LPC) coefficients (both the MFCC and LPC values are commonly used in MIR and Natural Language Processing systems). MFCCs are well suited to MIR tasks because they have been designed specifically to represent features which the human auditory response system perceives [22]. LPC is a compression algorithm which represents the $n$-th sample of a signal using a linear combination of the $p$ previous samples combined with an error correction term [24]. The algorithm is based on a simple model of the human vocal system.

Before feature extraction, the audio signals for songs and queries are normalized so that differences in sound levels are negligible. Because a user querying for a part of a song will be likely to preserve volume differences only within the query and not across the entire song, only local volume changes are important. Normalizing the signal preserves local volume changes while preventing volume discrepancies between songs and queries from affecting the results. All features were extracted for every window of 100ms with 50% overlap. As with the features selected, parameters were chosen due to their popularity in other systems.

In addition to evaluating SOMs created for each user's dataset individually, stratified sampling was used to test the cross-user robustness of the system. Since multiple users choose to query by humming, whistling, and singing words, for each of those query styles, 5-fold cross validation was performed using data from all users with that query style. To create the five groups, each user's dataset was randomly divided into five groups and then the groups for each user were combined. So each user's dataset was spread equally across the five groups; users with larger datasets had more songs in each group than users with smaller datasets. As a final evaluation of the cross-user robustness of the system, we also performed 5-fold cross validation over all the users regardless of query style. The groups were created as above so that each user's dataset was spread equally over the five groups. The cross-user experiments used only the MFCCs for features. All of the SOMs trained were 128x128 in size, as preliminary tests suggested this size would provide a balance between training time and system performance.

### C. Results

In the overall results for each user, the average value of each of the metrics (Equations 2-5) across the five folds is compared to the the expected value of the metric when using randomly generated orderings (Equations 6-9). This comparison is reported using the percent improvement,

$$\iota = \frac{\alpha - \upsilon}{\upsilon} \times 100 \tag{10}$$

where $\iota$ is the percent improvement, $\alpha$ is the average value of the metric, and $\upsilon$ is the expected value of the metric when using randomly generated orderings.

For the average position metric, lower results are better so the percent improvement equation is modified to reflect this:

$$\iota = \frac{\upsilon - \alpha}{\upsilon} \times 100 \tag{11}$$

This makes all of the percent improvement values to be comparable: higher is better. It is important to remember when viewing results in this section that percent improvement is very sensitive when the values are near zero.

*1) Single user, MFCC:* For the first experiment, each dataset (user) was treated independently, and the only features extracted were the MFCCs. Overall there is definite improvement over the expected value of random orderings (Table II). The SOMs trained for users A, C, and E all had 100% accuracy on all five folds over their respective training sets. The SOMs trained for user F averaged 86.15% correct with 2 folds reaching 100%. This result is most likely because these four users' training sets were relatively small—the large size of the SOM compared to the number of queries simplifies the task. This success on the training sets does seem to transfer to the test sets, though not as strongly as we would like. The standard deviations of the metrics are rather large compared to their means. This suggests that the result of training a SOM varies from user to user. Despite the large standard deviations, the average across users of each of the metrics shows definite improvement over random orderings.

As just mentioned, the size of the dataset relative to the size of the SOM has a strong effect on how effective the SOM is at separating different data while grouping similar data. For performance on the training set, there is an inverse correlation between average fold size and improvement in average position (see "Training Set/Position" column in Table II). Clearly the SOM size compared to the fold size is the dominating factor in how effective our system is in this case. However, for the test sets, the correlation disappears—the size of the SOM affects its ability to recall specific instances but does not directly impact its generalization.

*2) Single user, MFCC and LPC:* Next, the LPC coefficients were added to the feature set in an attempt to improve performance. The results (Table III) show that, for the training set, including LPC coefficients was detrimental

| | | Training Set | | | | Test Set | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Fold Size (train/test) | Correct | Top 5 | Top 10 | Position | Correct | Top 5 | Top 10 | Position |
| User E | 5.2/20.8 | 420.02 | 4.00 | 0.00 | 100.00 | 21.83 | 4.20 | -6.01 | 1.15 |
| User A | 7.0/28.0 | 600.28 | 40.00 | 0.00 | 100.00 | 100.00 | 28.00 | 18.01 | 5.63 |
| User C | 10.6/42.4 | 960.45 | 112.00 | 6.00 | 100.00 | 0.00 | 0.00 | 6.11 | 4.20 |
| User F | 12.6/50.4 | 985.01 | 152.02 | 26.01 | 96.72 | -40.40 | 3.83 | 6.00 | 1.21 |
| User H | 22.4/89.6 | 598.21 | 199.64 | 85.84 | 57.20 | -20.54 | 15.95 | 0.00 | 0.52 |
| User G | 23.0/92.0 | 339.77 | 143.97 | 80.03 | 47.36 | 0.00 | 16.02 | 18.03 | -0.37 |
| User B | 29.6/118.4 | 316.27 | 139.67 | 68.09 | 35.59 | 60.71 | 16.11 | 20.00 | 3.70 |
| User D | 42.0/168.0 | 39.92 | 32.02 | 26.00 | 10.54 | 18.33 | -4.36 | 22.02 | 1.99 |
| $\mu$ | 19/76.2 | 532.49 | 102.92 | 36.50 | 68.43 | 17.49 | 7.97 | 9.63 | 2.25 |
| $\sigma$ | 11.8/47.2 | 324.26 | 69.34 | 36.14 | 35.44 | 44.89 | 10.76 | 9.97 | 2.05 |

TABLE II
*Single user MFCC percent improvement.* $\mu$ AND $\sigma$ ARE THE MEAN AND STANDARD DEVIATION RESPECTIVELY.

(compare to Table II). However, the percentage correct and percentage in the top 5 improved overall in the test sets. Users B and H showed improvement on the test sets for all the metrics compared to using only MFCCs. User G improved on the test sets for all the metrics except percentage in the top 10. The results for user C were mixed, improving on the first two metrics and getting worse for the last two. Users A, D, E, and F, however, showed an overall decrease in performance. Overall there was improvement on the test sets on the first two metrics, but decreased performance on the last two. This mixed improvement comes at the cost of substantial increases in the standard deviations for all but the first metric—the effect of including LPC coefficients in the feature set varies with the user. For the training sets, the inverse correlation between training set size and percent improvement of average position still exists, though it is much less pronounced compared to when only MFCCs are used, and, again, disappears in the test sets.

*3) Cross User, Single Style:* For three query styles (humming, whistling, and singing words) there was more than one test user. Three users queried by singing words, two users queried by whistling, and two users queried by humming. To explore the robustness of the system, 5-fold cross validation was performed for each of those query styles. MFCCs were the only features used in these tests. The results suggest that the system can learn the training set, but is ineffective at generalizing to the test data (see Table IV). The results for the humming style do show improvement for three of the four metrics on the test set and only little improvement in the training set. This suggests that the training did result in a more general model and not just superficial memorization of training patterns (as experiments on the other two styles suggest happened). This may be due to the fact that the users that hummed provided the largest datasets, and in this combined experiment had an average fold size of 71.60 query-target pairs—the larger training set allows the SOM to learn a more general relationship.

*4) Cross User, Cross Style:* An ideal solution to musical query-by-content would be robust across different users and query styles. The results of a stratified sampling test combining data from all eight subjects are presented in Table V. As in the single style experiment, the system shows improvement on the training set. The test set results show slight improvement over random orderings for the average position metric. The improvement in the percent correct metric in Table V is misleading because the values are close to zero and one fold showed enough improvement to skew the average, whereas the average of the other four folds would be exactly the expected value of random orderings. The trained SOM is not able to generalize to the test sets in this experiment, but this is perhaps to be expected given the variety of query styles and paucity of training data.

*5) Comparison to Other Systems:* It is desirable to compare these results with those of other query-by-content systems. Two issues to consider when making such a comparison are that other query-by-content systems restrict the query options of the user to a single style and that these systems have been in development for over 15 years, allowing many improvements to have been made to initial designs. The system presented here, however, is the first we know of that allows the user to query in any style they choose. Therefore, it should be expected that the more restricted and mature systems will yield better results than this new system. In addition to these issues, the amount of testing done on other systems appears to be minimal and provides very little information with which to compare our results. Of the three systems that provided experimental results, Lu, You, and Zhang tested an unspecified number of users with a total of 42 queries from 1000 songs [25]; Liu and Tsai tested four users with 10 songs each [20]; and Raju, Sundaram, and Rao tested five users with 20 songs each [26]. In comparison, we tested eight users making 762 queries on 783 songs.

Lu, You, and Zhang report that 88% of queries produced a result within the top 10 results, which is a 270% improvement over the expected value of random orderings [25]. Liu

|  |  | Training Set |  |  |  | Test Set |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
|  | Fold Size (train/test) | Correct | Top 5 | Top 10 | Position | Correct | Top 5 | Top 10 | Position |
| User E | 5.2/20.8 | 97.61 | 0.54 | 0.00 | 26.67 | -38.67 | -7.49 | 6.07 | 0.61 |
| User A | 7.0/28.0 | 100.07 | 12.00 | 0.00 | 25.67 | -19.89 | -20.04 | 4.00 | 1.11 |
| User C | 10.6/42.4 | 40.72 | 24.13 | 2.14 | 13.96 | 39.83 | 7.71 | -6.02 | 1.59 |
| User F | 12.6/50.4 | -19.27 | 12.12 | 6.14 | 2.59 | 0.00 | -3.93 | -10.03 | -0.93 |
| User H | 22.4/89.6 | 39.91 | 24.33 | 4.12 | 6.82 | 0.00 | 15.95 | 24.01 | 5.19 |
| User G | 23.0/92.0 | 20.00 | 0.00 | 0.00 | 2.18 | 39.45 | 36.10 | 5.98 | 0.79 |
| User B | 29.6/118.4 | 19.53 | 32.37 | 32.09 | 11.19 | 101.19 | 56.16 | 32.11 | 5.20 |
| User D | 42.0/168.0 | 0.00 | 4.03 | -2.02 | -0.54 | 38.33 | -4.03 | -17.98 | -0.42 |
| $\mu$ | 19/76.2 | 37.32 | 13.69 | 5.31 | 11.07 | 20.03 | 10.05 | 4.77 | 1.64 |
| $\sigma$ | 11.8/47.2 | 42.77 | 12.13 | 11.13 | 10.48 | 43.97 | 25.22 | 16.79 | 2.34 |

TABLE III

*Single user MFCC & LPC percent improvement.* $\mu$ AND $\sigma$ ARE THE MEAN AND STANDARD DEVIATION RESPECTIVELY.

|  |  | Training Set |  |  |  | Test Set |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
| Style | Fold Size (train/test) | Correct | Top 5 | Top 10 | Position | Correct | Top 5 | Top 10 | Position |
| Whistling | 35.6/142.4 | 240.21 | 80.41 | 49.98 | 17.11 | 0.00 | 7.98 | -7.98 | -1.22 |
| Singing | 40/160 | 438.40 | 155.52 | 85.80 | 35.79 | -20.00 | -4.00 | -11.68 | -0.63 |
| Humming | 71.6/286.4 | 0.00 | 0.00 | 14.03 | 6.32 | -60.00 | 4.60 | 18.05 | 2.43 |

TABLE IV

*Cross user, single style MFCC percent improvement.* EACH SOM WAS TRAINED ON A STRATIFIED SAMPLE ACROSS ALL USERS OF EACH QUERY STYLE.

| Training Set |  |  |  | Test Set |  |  |  |
|---|---|---|---|---|---|---|---|
| Correct | Top 5 | Top 10 | Position | Correct | Top 5 | Top 10 | Position |
| 140.91 | 40.55 | 42.38 | 5.80 | 62.50 | -15.85 | -1.83 | 1.98 |

TABLE V

*Cross user, cross style MFCC percent improvement.* EACH SOM WAS TRAINED ON A STRATIFIED SAMPLE ACROSS ALL USERS AND QUERY STYLES.

and Tsai did not provide enough detail for us to be able to compare their results to our own. Raju, Sundaram, and Rao report that 95% of queries returned the correct result first, an 1800% improvement over the expected value of random orderings [26]. These results are, as expected, better than ours; however, our experiments are more comprehensive and our system more robust to query style. Given the inherent differences between other, more limited query-by-content systems and our more general system, as well as the limited results provided for those systems, we consider our comparison against random orderings to be a fair and reproducible way of evaluating our system.

## V. Conclusion

Musical query-by-content systems, like the one introduced here, allow users to find music even when they cannot remember the artist, title, or lyrics. Most other current query-by-content systems reduce the information-rich audio signal to a string representation of only the melody; queries are similarly processed and compared using string matching techniques. Many of these systems also require that the user query in a certain style. These restrictions reduce the flexibility of the system and force the user to interact in a specific way which may be unnatural. Our system does not have these restrictions. Using a Self-Organizing Map, we are able to create a query-by-content system which is independent of the query style of the user and is able to retain more of the acoustical content of the audio signal. This avoids the difficult task of automatically extracting a single melody and representing it in some simplified form.

Our results show that it is feasible to use a Self-Organizing Map in a query-by-content system. The system is most effective when using a dataset from a single user with a single query style. However, unlike most other musical query-by-content systems, our system does not dictate the style of the query. The user may query in whatever style is most comfortable and the system is able to remain effective. This is an important step forward in making query-by-content systems more intuitive and useful to users.

As far as we have been able to determine, this work is the first to use Self-Organizing Maps in a musical query-by-content system. Our work shows that SOMs are a viable option for creating a query-by-content system, but there is still room for improvement.

The size of the SOM relative to the size of its training set is important for determining the SOM's efficacy. We used a fixed size that we felt balanced training time with performance. The size of the SOM can be determined in a more online fashion by using a hierarchical SOM, a self-growing SOM, or both. These automatically sizing SOMs allow the system to more easily adjust to the amount of data available. Of course, this would incur an associated increase in training time; however, the trade-off is the reduced query time resulting from these online SOMs—an important factor in developing a scalable system.

Our approach would require modification in order to effectively scale to production-size datasets. While improving the query time of the SOM is important, the bulk of the computation occurs when comparing the computed path to the stored paths for all songs in the library. One approach to this problem was presented by Cui et al. which involves storing signatures in a database for quick elimination of unlikely candidates [27]. Additionally, the system currently does not handle varying tempo in queries. A possible approach would be to interpolate audio samples or remove samples in order to create longer or shorter queries.

The results suggest that larger training sets allow the SOM to learn a more general relationship. In each fold we used only 20% of the dataset for training; simply using more of the data in training would likely increase performance.

The overall cross-user, cross-style robustness of the system might be improved by training on a dataset where each user provides query samples for the same set of targets or by augmenting SOM vectors with multiple query vectors of different styles. To better analyze the suitability of this type of system, a larger study should be conducted including more users, larger datasets, datasets which match multiple users' queries to the same target, and more varied query styles.

## References

[1] N. Scaringella, G. Zoia, and D. Mlynek, "Automatic genre classification of music content: a survey," *Signal Processing Magazine, IEEE*, vol. 23, no. 2, pp. 133–141, 2006.

[2] N. Kosugi, Y. Nishihara, T. Sakata, M. Yamamuro, and K. Kushima, "A practical query-by-humming system for a large music database," in *Proceedings of ACM International Conference on Multimedia*. New York, NY, USA: ACM Press, 2000, pp. 333–342.

[3] S. Pauws, "Cubyhum: A fully operational "query by humming" system." in *Proceedings of International Conference on Music Information Retrieval*, Paris, France, October 2002.

[4] W. Birmingham, R. Dannenberg, and B. Pardo, "Query by humming with the vocalsearch system," *Communications of the ACM*, vol. 49, no. 8, pp. 49–52, 2006.

[5] K. Jacobson, "A multifaceted approach to music similarity," in *Proceedings of International Conference on Music Information Retrieval*, Victoria, Canada, October 2006, pp. 346–348.

[6] T. Pohle, E. Pampalk, and G. Widmer, "Evaluation of frequently used audio features for classification of music into perceptual categories," in *Proceedings of International Workshop on Content-Based Multimedia Indexing*, Riga, Latvia, June 2005.

[7] A. Meng and J. Shawe-Taylor, "An investigation of feature models for music genre classification using the support vector classifier," in *Proceedings of International Conference on Music Information Retrieval*, London, UK, September 2005, pp. 604–609, final version : 6 pages instead of original 8 due to poster presentation.

[8] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Transactions on Speech and Audio Processing*, vol. 10, no. 5, pp. 293–302, July 2002.

[9] E. Allamanche, J. Herre, O. Hellmuth, T. Kastner, and C. Ertel, "A multiple feature model for musical similarity retrieval," in *Proceedings of International Conference on Music Information Retrieval*, Baltimore, MD, USA, October 2003, pp. 217–218.

[10] J. Paulus and A. Klapuri, "Measuring the similarity of rhythmic patterns," in *Proceedings of International Conference on Music Information Retrieval*, M. Fingerhut, Ed., Paris, France, Oct 2002, pp. 150–156.

[11] B. Logan and A. Salomon, "A music similarity function based on signal analysis," in *Proceedings of IEEE International Conference on Multimedia and Expo*, Tokyo, Japan, August 2001, pp. 745–748.

[12] J. Foote, M. Cooper, and U. Nam, "Audio retrieval by rhythmic similarity," in *Proceedings of International Conference on Music Information Retrieval*, Paris, France, October 2002.

[13] K. B. Dickerson and D. Ventura, "Music recommendation and query-by-content using self-organizing maps," in *Proceedings of the International Joint Conference on Neural Networks*, Atlanta, GA, USA, June 2009, pp. 705–710.

[14] T. Kohonen, *Self-Organizing Maps*. Springer, 2001.

[15] B. Feiten and S. Günzel, "Automatic indexing of a sound database using self-organizing neural nets," *Computer Music Journal*, vol. 18, no. 3, pp. 53–65, 1994.

[16] S. Harford, "Automatic segmentation, learning and retrieval of melodies using a self-organizing neural network," in *Proceedings of International Conference on Music Information Retrieval*, Baltimore, MD, USA, 2003.

[17] M. Dittenbach, D. Merkl, and A. Rauber, "The growing hierarchical self-organizing map," in *Proceedings of International Joint Conference on Neural Networks*, vol. 6. Washington, DC, USA: IEEE Computer Society, July 2000, p. 6015.

[18] A. Rauber, E. Pampalk, and D. Merkl, "Using psycho-acoustic models and self-organizing maps to create a hierarchical structuring of music by sound similarities," in *Proceedings of International Conference on Music Information Retrieval*, Paris, France, October 2002.

[19] J. Laaksonen, M. Koskela, and E. Oja, "Content-based image retrieval using self-organizing maps," in *Proceedings of International Conference on Visual Information and Information Systems*. London, UK: Springer-Verlag, 1999, pp. 541–548.

[20] C.-C. Liu and P.-J. Tsai, "Content-based retrieval of mp3 music objects," in *Proceedings of International Conference on Information and Knowledge Management*. New York, NY, USA: ACM, 2001, pp. 506–511.

[21] B. Cui, L. Liu, C. Pu, J. Shen, and K.-L. Tan, "Quest: querying music databases by acoustic and textual features," in *MULTIMEDIA '07: Proceedings of the 15th international conference on Multimedia*. New York, NY, USA: ACM, 2007, pp. 1055–1064.

[22] B. Logan, "Mel frequency cepstral coefficients for music modeling," in *In International Symposium on Music Information Retrieval*, Plymouth, MA, USA, October 2000, pp. 23–25.

[23] M. I. Mandel and D. P. W. Ellis, "Song-level features and support vector machines for music classification," in *Proceedings of International Conference on Music Information Retrieval*, London, UK, September 2005, pp. 594–599.

[24] D. Rocchesso, *Introduction to Sound Processing*. Davide Rocchesso, 2003.

[25] L. Lu, H. You, and H.-J. Zhang, "A new approach to query by humming in music retrieval," in *Proceedings of IEEE International Conference on Multimedia and Expo*, Tokyo, Japan, August 2001, pp. 595–598.

[26] M. A. Raju, B. Sundaram, and P. Rao, "Tansen: A query-by-humming based music retrieval system," in *Proceedings of Indian Institute of Technology National Conference on Communications*, Chennai, India, February 2003.

[27] B. Cui, H. V. Jagadish, B. C. Ooi, and K.-L. Tan, "Compacting music signatures for efficient music retrieval," in *EDBT '08: Proceedings of the 11th international conference on extending database technology*. New York, NY, USA: ACM, 2008, pp. 229–240.