# A QUERY BY HUMMING SYSTEM THAT LEARNS FROM EXPERIENCE

**David Little, David Raffensperger, Bryan Pardo**
EECS Department
Northwestern University
Evanston, IL 60201
d-little,d-raffensperger,pardo@northwestern.edu

## ABSTRACT

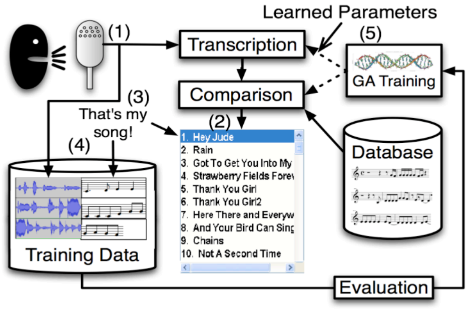
Query-by-Humming (QBH) systems transcribe a sung or hummed query and search for related musical themes in a database, returning the most similar themes. Since it is not possible to predict all individual singer profiles before system deployment, a robust QBH system should be able to adapt to different singers after deployment. Currently deployed systems do not have this capability. We describe a new QBH system that learns from user provided feedback on the search results, letting the system improve while deployed, after only a few queries. This is made possible by a trainable note segmentation system, an easily parameterized singer error model and a straight-forward genetic algorithm. Results show significant improvement in performance given only ten example queries from a particular user.

## 1. INTRODUCTION

Deployed search engines used to find music documents, such as amazon.com, rely on metadata about the song title and performer name as their indexing mechanism. Often, a person is able to sing a portion of the piece, but cannot specify the title, composer or performer. Query by humming (QBH) systems [1] solve this mismatch between database keys and user knowledge by matching a sung query to musical themes in a database, returning the most similar themes.

One of the main difficulties in building an effective QBH system is dealing with the variation between sung queries and the melodies used as database search keys. Singers may go out of tune, sing at a different tempo than expected, or in a different key [1, 7]. Further, singers differ in their error profiles. One may have poor pitch, while another has poor rhythm.

Since it is not possible to predict all individual singer profiles before deployment, a robust QBH system should be able to adapt to different singers after deployment. Current QBH systems do not have this capability. While there has been significant prior work that addresses (or is applicable to) singer error modelling [7, 9, 11] for QBH, researchers have not focused on fully automated, ongoing QBH optimization after deployment. Thus, these approaches are unsuited for this task, requiring either hundreds of example queries to customize to an individual [7, 9], or training examples where the internal structure of each query is aligned by the trainer to the structure of the target [11].



1. Hey Jude
2. Rain
3. Got To Get You Into My
4. Strawberry Fields Forev
5. Thank You Girl
6. Thank You Girl2
7. Here There and Everyw
8. And Your Bird Can Sing
9. Chains
10. Not A Second Time

**Figure 1.** System diagram

We are developing a QBH system (Figure 1) that personalizes a singer model based on user feedback, learning the model on-line, after deployment without intervention from the system developers and after only a few example queries. The user sings a query (step 1 in the figure). The system returns a list of songs from the database, ranked by similarity (step 2). The user listens to the songs returned and selects the desired one (step 3). The more a person uses and corrects the system, the better the system performs. Our system employs user feedback to build a database of paired queries and correct targets (step 4). These pairings are used to optimize the parameters of our note segmentation and note interval similarity parameters for specific users (step 5) or groups of users.

In this paper, we focus on how we automatically optimize backend QBH system performance, given a small set of example queries. We refer the reader to [10] for a description of the user interface and user interaction.

## 2. QUERY REPRESENTATION

In a typical QBH system, a query is first transcribed into a time-frequency representation where the fundamental frequency and amplitude of the audio is estimated at very short fixed intervals (on the order of 10 milliseconds). We call this sequence of fixed-frame estimates of fundamental frequency a *melodic contour representation*. Figure 2 shows the melodic contour of a sung query as a dotted line.

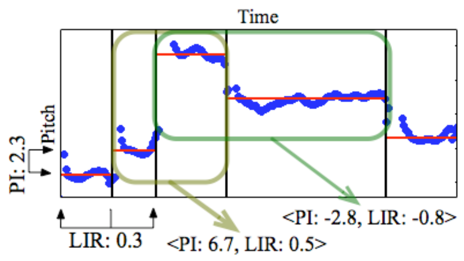


**Figure 2.** Example note intervals as <PI, ,LIR> pairs.

We segment the melodic contour into notes and then use a note interval representation. The pitch of each note is the median value in its segment. Each note interval is represented by the *pitch interval* (PI) between adjacent note segments (encoded as un-quantized musical half-steps) and the log of the ratio between the length of a note segment and the length of the following segment (LIR) [8]. Figure 2 shows several note intervals as PI, LIR pairs.

This representation has several advantages over a melodic contour: it is both transposition and tempo invariant. It is also compact, only encoding salient points of change (note transitions), rather than every 10 millisecond frame. This results in a speed-up of two orders of magnitude when matching queries to targets. Work in [1] has shown that the precision and recall of search using *quantized* note intervals is slightly worse than when using melodic contour.

We use *unquantized* note intervals. Use of unquantized PI and LIR values makes the representation insensitive to issues caused by a singer inadvertently singing in an unexpected tuning (A4 $\neq$ 440), or slowly changing tuning and tempo over the course of a query. This improves search performance relative to quantized note intervals. A previous study showed that use of unquantized note intervals significantly improved search performance compared to quantized note intervals [5].

## 3. NOTE SEGMENTATION

Our system first transcribes the query as a sequence of 10 millisecond frames. Each frame is a three element vector containing values for pitch, amplitude and harmonicity (relative strength of harmonic components to non harmonic components) [13].

We assume significant changes in these three features occur at note boundaries. Thus, we wish to determine what constitutes significant change. For example, a singer may use vibrato at times and not at other times. Thus, the amount of local pitch variation that constitutes a meaningful note boundary in one query may be insufficient to qualify as a note boundary in another query by the same singer. We wish to take local variance into account when determining whether or not a note boundary has occurred.

Note segmentation is related to the problem of visual edge detection [3]. Accounting for local variation has been helped edge detection in cases where portions of the image may be blurry and other portions are sharp [3]. The Mahalanobis distance [6] differs from the Euclidean distance in that it normalizes distances over a covariance matrix *M*. Using the Mahalanobis lets one measure distance between frames relative to local variation. In a region of large variance, a sudden change will mean less than in a relatively stable region. A previous study showed that our use of the Mahalanobis over Euclidean distance significantly improved search performance [5]

We find the distance between adjacent frames in the sequence using the Mahalanobis distance measure, shown in Equation 1. Given a frame $\mathbf{f}_i$, we assume a new note has begun wherever the distance between two adjacent frames $\mathbf{f}_i$ and $\mathbf{f}_{i+1}$, exceeds a threshold, *T*.

$$\sqrt{(\mathbf{f}_i - \mathbf{f}_{i+1})M^{-1}(\mathbf{f}_i - \mathbf{f}_{i+1})'} > T \Rightarrow \text{new note} \qquad (1)$$

The matrix *M* is a covariance matrix, calculated from the variance within a rectangular window around the frame $\mathbf{f}_i$.

Our note segmenter has four tuneable parameters: the segmentation threshold (*T*), and the weights (*w*) for each of the three features (pitch, harmonicity and amplitude). We address tuning of these four parameters in Section 6.

Once we have estimated note segment boundaries, we build note intervals from these note segments.

## 4. MODELING SINGER ERROR

Once a query is encoded as a sequence of note intervals, we compare it to the melodies in our database. Each database melody is scored for similarity to the query using a dynamic-programming approach to performing string alignment [9]. Rather than use a fixed match reward, the match reward is based on a similarity function *s* for note intervals. Ideally we would like interval $a_i$ to be similar to interval $b_j$ if $a_i$ likely to be sung when a singer intended to sing $b_j$. That is, likely errors should be considered similar to the correct interval, and unlikely errors should be less similar. Such a function lets a string-alignment algorithm correctly match error-prone singing to the correct target, as long as the singer is relatively consistent with the kinds of errors produced.

In previous work [9], we had participants listen to note intervals and attempt reproduce the intervals by singing. This study showed that the most common errors were octave displacements of one or two octaves. The next most common errors were half-step and whole step errors around the expected note interval, or around peaks offset by an octave. This supports the observations of Shepard [12], who proposes a pitch chroma representation where octaves are relatively close to each other in the chroma space.

This suggests that singing errors can be effectively modelled by a set of Gaussian distributions centered on the expected pitch interval and on intervals offset by one or more octaves. The normal function, $N(a,\mu,\sigma)$ returns the value for *a* given by a Gaussian function, centered on $\mu$, with a standard deviation σ. Equation 4 shows our note-interval similarity function, based on the normal function.

$$s(x,y) = w_r N(y_r, x_r, \sigma_r) + w_p \sum_{i=-n}^{n} \lambda^{|i|} N(y_p, x_p + 12i, \sigma_p) \quad (4)$$

Let $x$ and $y$ be two note intervals. Here, $x_p$ and $y_p$ are the pitch intervals of $x$ and $y$ respectively, and $x_r$ and $y_r$ are the rhythmic ratios (LIRs) of $x$ and $y$. The values $w_p$ and $w_r$ are the weights assigned to pitch and rhythm. The sum of $w_p$ and $w_r$ is 1.

The pitch similarity is modeled using $2\underline{n}+1$ Gaussians, each centered at one or more octaves above or below the expected pitch interval. The height of each Gaussian is determined by an octave decay parameter λ, in the range from than 1 to 0. This similarity function provides us with five parameters to tune: the pitch and rhythm weight ($w_p$ and $w_r$), the sensitivity to distances for pitch and rhythm ($\sigma_p$ and $\sigma_r$), and the octave decay ($\lambda$). Figure 3 shows two octaves of the positive portion of the pitch dimension of this function, given two example parameter settings.

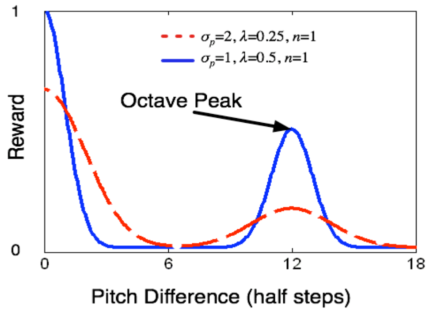


**Figure 3.** The pitch dimension of the similarity function in Equation 5.

## 5. SYSTEM TRAINING

We train the system by tuning the parameters of our note segmenter (Equations 1 and 2) and note similarity reward function (Equation 5). We measure improvement using the *mean reciprocal rank* (MRR) of a set of *n* queries. We define the *rank* of a query as the order of the correct song in the search results. MRR emphasizes the importance of placing correct target songs near the top of the list while still rewarding improved rankings lower down on the returned list of songs [1]. Values for MRR range from 1 to 0, with higher numbers indicating better performance. A MRR of 0.25 indicates the correct answer was, on average, in the top four songs returned by the search engine.

We use a simple genetic algorithm [14] to tune system parameters. Each individual in the population is one set of parameter values for Equations 1, 2 and 5. The fitness function is the MRR of the parameter settings over a set of queries. The genetic algorithm represents each parameter as a binary fraction of 7 bits, scaled to a range of 0 to 1. We allow crossover to occur between (not within) parameters.

During each generation, the fitness of an individual is found based on the MRR of the correct targets for a set of queries. Parameter settings (individuals) with high MRR values are given higher probability of reproduction (fitness proportional reproduction).

## 6. EMPIRICAL EVALUATION

Our empirical evaluation sought to evaluate the extent to which the system was able to improve search performance in response to training, both in the case of individualized training to a particular singer and also general training over a larger set of singers.

Our query set was drawn from the QBSH corpus [4] used during the 2006 MIREX comparison of query-by-humming systems [2]. We used 10 singers, each singing the same 15 songs from this dataset. Our target database was composed of the 15 targets corresponding to these queries plus 986 distracter melodies drawn from a selection of Beatles songs, folk songs and classical music, resulting in a database of 1001 melodies. Chance performance, on a database of this size would result in an MRR ≈ 0.005, given a uniform distribution.

For the genetic algorithm, we chose a population size of 60. Initial tests showed learning on this task typically ceases by the 30th generation, thus results shown here report values from training runs of 40 generations.

In practice, we would like to utilize user-specific training only when it improves performance relative to an un-personalized system. One simple option is to only use user-specific parameters if the user-specific performance ($MRR_u$) is superior to the performance using parameters learned on a general set of queries by multiple users ($MRR_g$).

To test this idea, we first trained the system on all queries from nine of ten singers. We then tested on all the queries from the missing singer. Cross validation across singers was performed, thus the experiment was repeated ten times, testing with the queries from a different singer each time. To speed learning, training was done using a random sample of 250 target songs from the database. For each trial, the set of parameters with the best training performance was evaluated by finding the MRR of the testing queries, searching over all 1001 melodies in the database. This gave us parameters for each singer that were learned on the queries by the other nine singers. These are the *general* parameter settings for a singer. The mean MRR testing performance of the *general* parameters was 0.235 (Std. Dev.=0.063).

We then performed a user-specific version of training. We used 3-fold cross validation across 15 queries for each of the same ten singers used for training the *general* parameters: we optimized parameters on the selected ten queries and tested on the remaining five. This provided us with 30 total trails for the *specific* parameters: three trials for each of the ten singers. The mean MRR testing performance for the *specific* parameters was: 0.228 (Std Dev. = 0.14).

For each trial we compared $MRR_s$ (the training performance of the learned user-specific parameters) to $MRR_g$ (the training performance of the general parameters learned from the other nine singers). If $MRR_s > MRR_g + \varepsilon$ on the training set, we used the user-specific parameters. Else, we used the general parameters. For this experiment, $\varepsilon$ was an error margin set to 0.04.

Once the parameters (general or user-specific) were selected, we tested them on the testing set for that trial. We called this a *combined* trial. The combined trials had an average MRR of 0.289 (Std. Dev. = 0.086). A t-test indicated the improvement of the *combined* results over the *general* and the *specific* parameter settings is statistically significant ($p \leq 0.024$).

On 50% of the combined trials the *specific* parameters were used and improved performance compared to *general* parameters. On 13% of the trials, *specific* parameters were used, but had worse testing performance than the *general* parameters. On the remaining 36% of trials, the *general* parameters were used. Figure 4 shows the average MRR performance for untrained, general and combined parameters.

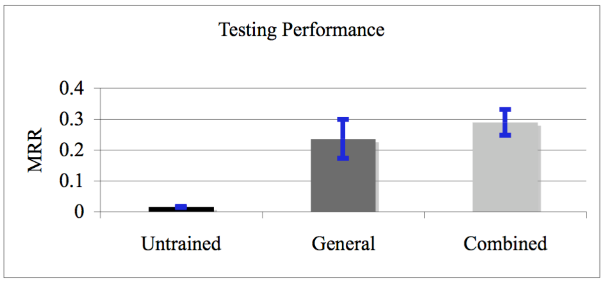


**Figure 4.** Average search performance using untrained, *general* and *combined* (both User-Specific and General) parameters.

## 7. CONCLUSIONS

We have described a QBH system that automatically customizes parameters to individuals or groups after deployment. Our results show that by correctly combing parameters trained to specific users, and a set trained over a general population, these combined parameters significantly improve mean search performance, resulting in a mean MRR of 0.289 on a database of 1001 melodies. This roughly corresponds to consistently placing the correct target in the top four results. This compares to an MRR of 0.0151 (Std. Dev. = 0.0018) prior to training. Our results also show unquantized note intervals and note segmentation that takes into account local pitch variation significantly improve performance.

In future work we will explore how performance varies with respect to the number of training examples and improve the information that can be used for training while maintaining user-specificity. We will also explore more sophisticated criteria to determine when user-specific training should be used.

## 8. REFERENCES

[1] R. Dannenberg, W. Birmingham, B. Pardo, N. Hu, C. Meek and G. Tzanetakis, "A Comparative Evaluation of Search Techniques for Query-by-Humming Using the MUSART Testbed", *Journal of the American Society for Information Science and Technology* (2007), pp. in press.

[2] J. S. Downie, K. West, A. Ehmann and E. Vincent, "The 2005 Music Information retrieval Evaluation Exchange (MIREX 2005): Preliminary Overview", *6th International Conference on Music Information Retrieval*, September 11-15, London, UK, 2005.

[3] J. H. Elder and S. W. Zucker, "Local scale control for edge detection and blur estimation", *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20 (1998), pp. 699-716.

[4] Jyh-Shing and R. Jang, "QBSH: A corups for Designing QBSH (Query by Singing/Humming) Systems", 2006.

[5] D. Little, D. Raffensperger and B. Pardo, "Online Training of a Music Search Engine", Northwestern University, Evanston, IL, NWU-EECS-07-03, 2007

[6] P. C. Mahalanobis, "On the generalised distance in statistics, " *Proceedings of the National Institute of Science of India* 12 (1936), pp. 49-55.

[7] C. Meek and W. Birmingham, "A Comprehensive Trainable Error model for sung music queries", *Journal of Artificial Intelligence Research*, 22 (2004), pp. 57-91.

[8] B. Pardo and W. Birmingham, "Encoding Timing Information for Music Query Matching", *International Conference on Music Information Retrieval*, Paris, France, 2002.

[9] B. Pardo, W. P. Birmingham and J. Shifrin, "Name that Tune: A Pilot Study in Finding a Melody from a Sung Query", *Journal of the American Society for Information Science and Technology*, 55 (2004), pp. 283-300.

[10] B. Pardo and D. Shamma, "Teaching a Music Search Engine through Play", *CHI 2007, Computer/Human Interaction* San Jose, California, 2007.

[11] C. Parker, A. Fern and P. Tadepalli, "Gradient boosting for sequence alignment", *The Twenty-First National Conference on Artificial Intelligence*, Boston, MA, 2006.

[12] R. N. Shepard, "Geometrical Approximations to the structure of musical pitch", *Psychological Review*, 89 (1982), pp. 305-309.

[13] G. Tzanetakis and F. Cook, "A framework for audio analysis based on classification and temporal segmentation", *EUROMICRO Conference*, Milan, 1999, pp. 61-67.

[14] A. Wright, "Genetic algorithms for real parameter optimization", *The First workshop on the Foundations of Genetic Algorithms and Classier Systems*, Bloomington, Indiana, 1990.