NORTHWESTERN
UNIVERSITY

Electrical Engineering and Computer Science Department

Technical Report
NWU-EECS-07-03
July 12th, 2006

Online Training of a Music Search Engine

David Little, David Raffensperger, and Bryan Pardo.

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 as a play list. A major obstacle to effective QBH is variation between user queries and the melodic targets used as database search keys. 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.

Keywords: Query-by-humming, user error modeling, user-specific training, genetic algorithms.

*A version of this paper was submitted to the 4th Joint Workshop on Multimodal Interaction and Related Machine Learning Algorithms.

1 Introduction

Most currently deployed music search engines, such as Amazon.com and local libraries, make use of metadata about the song title and performer name in 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 [1–5] solve this mismatch between database keys and user knowledge. This is done by transcribing a sung query and searching for related musical themes in a database, returning the most similar themes as a play list.

One of the main difficulties in building an effective QBH system is dealing with the variation between user queries and the melodic targets used as database search keys. Singers are error prone: they may go out of tune, sing at a different tempo than expected, or in a different key, and notes may be removed or added [1, 6]. Further, singers differ in their error profiles. One singer may have poor pitch, while another may have poor rhythm. Similarly, different environments may introduce variation through different levels of background noise or availability of different microphones.

Since it is not possible to predict all individual singer profiles or use cases before deployment of a system, a robust QBH system should be able to adapt to different singers and circumstances after deployment. Currently deployed QBH systems do not have this capability.

While there has been significant prior work that addresses (or is applicable to) singer error modelling [6–8, 5] 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 [6, 7], or training examples where the internal structure of each query is aligned to the structure of the correct target [8].

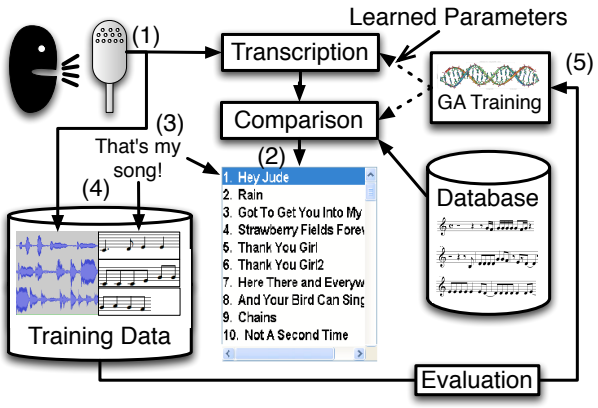

Fig. 1. System overview

We are developing a QBH system (shown in 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. To do this, our system employs user feedback to build a database of paired queries and correct targets (step 4). These pairings are then used to learn optimal note segmentation and note interval similarity parameters for specific users (step 5).

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 [9] for a description of the user interface and user interaction. Section 2 describes our query representation. Section 3 describes our optimizable note segmentation algorithm. Section 4 describes our melodic comparison algorithm. Section 5 describes our optimizable note interval similarity function. Section 6 describes our genetic algorithm learning approach. Section 7 outlines an empirical study of our system. Section 8 contains conclusions and future directions.

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.

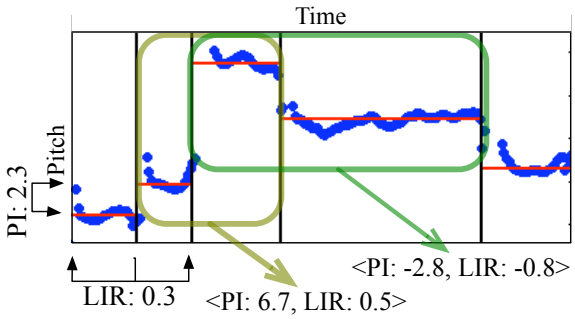

Fig. 2. Several unquantized pitch intervals built from a melodic contour.

Finding queries represented by melodic contour imposes computational overhead on the search engine [10]. The melodic contour representation uses absolute

frequency and tempo, making it sensitive to queries that vary in tempo and key from the target. To overcome these problems one can transform the contour through a number of keys and tempi [1] but this incurs significant overhead.

To improve system speed, we encode a query as a series of note intervals, created as follows:

1. Divide the melodic contour into segments corresponding to notes.
2. Find the median pitch and the length in frames of each note segment.
3. Encode segments as a sequence of note intervals.

Each note interval is represented by a pair of values: 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) [10]. Since each LIR is a ratio, values do not have units.

This representation is both transposition and tempo invariant. It is also compact, only encoding salient points of change (note transitions), rather than every 10 millisecond frame. Figure 2 shows several note intervals.

We note that the 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 lets us avoid having to model these common singer errors.

The note-interval representation speeds search by a factor of roughly 100, when compared with a melodic contour representation on real-world queries [1]. This speedup comes at the price of introducing potential errors when dividing the query into note segments. We address note segmentation in Section 3.

3 Note Segmentation

In the initial transcription step, our system estimates the pitch, root-mean-squared amplitude and harmonicity (the relative strength of harmonic components compared to non-harmonic components) of the audio. This is done every 10 milliseconds, resulting in a sequence of fixed-length frames, each of which is a vector of these three features.

We assume significant changes in these three features occur at note boundaries. Thus, we wish to find a good way to determine what constitutes significant change. Our initial approach was to use a Euclidean distance metric on the sequence of frames. In this approach, a new note was assumed to begin whenever the distance between adjacent frames exceeds a fixed threshold.

Unfortunately, this introduces error when a single fixed threshold is not appropriate. For example, a singer may use vibrato at times and not at other times. Thus, the local pitch variation that might constitute 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 [11, 12]. Accounting for local variation has been successfully used in image processing to perform edge detection in cases where portions of the image may be blurry and other portions are sharp [13, 11]. The Mahalanobis distance [14] 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.

We find the distance between adjacent frames in the sequence using a Mahalanobis distance measure, shown in Equation 1. Recall that each frame is a three element vector containing values for pitch (p), amplitude (a) and harmonicity (h). Given a frame f_i, $f_i = \langle p_i, a_i, h_i \rangle$, we assume a new note has begun wherever the distance between two adjacent frames f_i and f_{i+1}, exceeds a threshold, T. This is shown in Equation 1.

$$\sqrt{(\mathbf{f}_i - \mathbf{f}_{i+1})\, M^{-1}\, (\mathbf{f}_i - \mathbf{f}_{i+1})'} \tag{1}$$

Our covariance matrix M depends on local variation in the three features of a frame. The matrix has three rows and three columns, with each row, ρ, corresponding to one feature and each column, η, corresponding to one feature (so ρ, $\eta \in \{p_i, a_i, h_i\}$ for f_i). We calculate each element of M for a frame f_i using Equation 2:

$$M_{\rho\eta} = \frac{1}{2\tau} \sum_{k=i-\tau}^{i+\tau} \left(\frac{\rho_k - \bar{\rho}}{w_\rho} \right) \left(\frac{\eta_k - \bar{\eta}}{w_\eta} \right) \tag{2}$$

In Equation 2, the w terms are weighting parameters that adjust the importance of the features of the frame (pitch, harmonicity and amplitude). The parameter τ is the size of a window (measured in frames) surrounding the current frame. The window size determines the number of frames considered in finding local variation. The averages for ρ and η ($\bar{\rho}$ and $\bar{\eta}$) are calculated over this window.

Thus, Equation 2 finds the covariance between a pair of features: for example with $\rho = a$, and $\eta = h$, Equation 2 would find the covariance between the amplitude and harmonicity for frames $i - \tau$ to $i + \tau$.

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. We leave the tuning of parameter τ for future work, setting it to a value of 25 frames (250 milliseconds) to either side of the current frame i.

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

4 Measuring Melodic Similarity

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 classic dynamic-programming approach to performing string alignment [7]. The note interval matcher computes the similarity $Q(A, B)$ between two melodic sequences $A = a_1, a_2, \cdots a_m$ and $B = b_1, b_2, \cdots b_n$. by filling the matrix $Q = (q_{1\cdots m,1\cdots n})$. Each entry q_{ij} denotes the maximum melodic similarity between the two prefixes $a_1 \cdots ai$ and $b_h \cdots b_j$ where $1 \leq h \leq j$.

We use a standard calculation method for the algorithm, shown in Equation 3

$$q_{i,j} = max \begin{cases} 0 \\ q_{i-1,j-1} + s(a_i, b_i) \\ q_{i-1,j} - c_a \\ q_{i,j-1} - c_b \end{cases} \tag{3}$$

Here, $s(a_i, b_j)$ is the similarity reward for aligning note interval a_i to note interval b_j. We define $s(a_i, b_j)$ in Section 5. The costs for skipping a note interval from melody A or from melody B are given by c_a and c_b, respectively. Equation 3 is a local alignment method, so any portion of the query is allowed to match any portion of the target. The overall similarity is taken to be the maximum value in the matrix Q.

5 Modeling Singer Error

We now define the 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 examples to the correct targets, as long as the singer is relatively consistent with the kinds of errors produced.

In our previous work [7], the similarity function was represented with a table of 625 (25 by 25) values for 25 possible pitch intervals (from -12 to 12 half steps). The similarity between a perfect fifth (7 half steps) and a tritone (6 half steps) was determined by calculating the statistical likelihood of a person singing a perfect fifth when instructed to sing a tritone. The more likely this was, the higher the similarity. This required hours of directed training to learn all pairings in our table. This is something users outside the laboratory setting are unwilling to do.

For this work, we have developed a similarity function that captures singer variation by tuning only a few parameters, so that suitable values can be learned quickly. The normal function, $N(a, \mu, \sigma)$ returns the value for a given by a Gaussian function, centered on μ, with a standard deviation σ. Equation 4 shows a simple note-interval similarity function, based on the normal function.

$$s(x, y) = w_p N(y_p, x_p, \sigma_p) + w_r N(y_r, x_r, \sigma_r) \quad (4)$$

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 LIRs of x and y. The values w_p and w_r are the weights of pitch and rhythm. The sum of w_p and w_r is 1. Equation 4 is maximal when x and y are identical. As the difference between x and y grows, Equation 4 returns a value approaching 0.

Equation 4 assumes increasing distance between pitch intervals is equivalent to decreasing similarity between intervals. There is at least one way in which this is untrue for pitch: octaves. Shepard [15] proposes a pitch similarity measure that accounts for two dimensions: pitch chroma and pitch height. In this model, octaves are fairly close to each other. Criani [16] proposes a psychological model of pitch similarity based on the Pearson correlations of temporal codes which represent pitched stimuli. These correlations show strong peaks not only near like pitches, but also near pitches an octave apart. In previous work [10] we also found high likelihood of octave substitutions in sung queries. This suggests that, at a minimum, we should account for octave similarities in our measure of pitch similarity. We thus modify Equation 4 to give Equation 5.

$$s(x, y) = w_r N(y_r, x_r, \sigma_r) + w_p \sum_{i=-n}^{n} \lambda^{|i|} N(y_p, x_p + 12i, \sigma_p) \quad (5)$$

Here, the pitch similarity is modeled using $2n+1$ Gaussians, each one centered at one or more octaves above or below the pitch of x. The height of each Gaussian is determined by an octave decay parameter λ, in the range from than 1 to 0. This reward 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 (σ_p and σ_r), and the octave decay (λ). Figure 3 shows the positive portion of the pitch dimension of this function, given two example parameter settings, with two octaves shown.

6 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. This is shown in Equation 6. Here, we define the rank of the ith query, r_i as the rank returned by the search engine for the correct song in the database.

$$MRR = \frac{\sum_{i=1}^{n} \frac{1}{r_i}}{n} \quad (6)$$

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

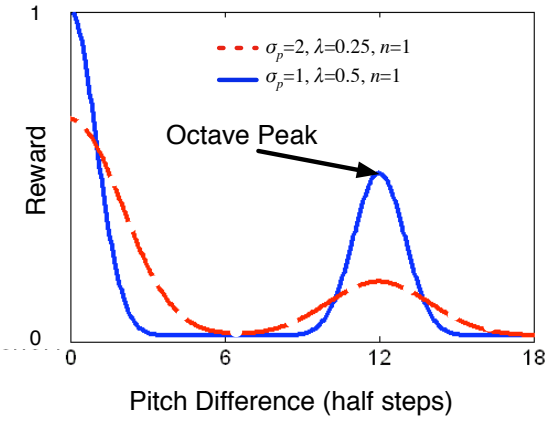

Fig. 3. The pitch dimension of the similarity function in Equation 5.

list of songs [1]. Values for MRR range from 1 to 0, with higher numbers indicating better performance. Thus, MRR = 0.25 roughly corresponds to the correct answer being in the top four songs returned by the search engine, MRR = 0.05 indicates the right answer is in the top twenty, and so on.

We use a simple genetic algorithm [17, 18] 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 Mean Reciprocal Rank (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). We speed training by finding MRR over random subset of the database.

Given ten queries from each singer labeled with the correct target name and a set of 250 target melodies from our database, the system personalizes two singers per hour on our current hardware. These results are obtained with an iMac 2.16 GHz Intel Core Duo 2, with 2 GB of RAM. At this speed, our system can update models overnight for up to 16 singers who have provided new queries during the previous day.

7 Empirical Evaluation

Our empirical evaluation sought to compare user-specific training to training on a general set of queries. We also sought to verify the utility of two new design

choices: the use of a Mahalanobis distance to account for local variation during note segmentation, and the use of unquantized, as opposed to quantized note-intervals.

Our query set was drawn from the QBSH corpus [19] used during the 2006 MIREX comparison of query-by-humming systems [20]. 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 1471 distracter melodies drawn from a selection of Beatles songs, folk songs and classical music, resulting in a database of 1486 melodies. Chance performance, on a database of this size would result in an $MRR \approx 0.005$, 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 30^{th} generation, thus results shown here report values from training runs of 40 generations. We used a mutation probability of 0.02 per parameter.

We ran two experiments: the first examines the overall performance of user specific training and the improvements the new system features introduced. The second compares our system with using user-specific to a version without user-specific training.

7.1 Experiment 1

For Experiment 1 we considered three different conditions. In the first condition we accounted for local variation during note segmentation using the Mahalanobis distance and used an unquantized note interval representation (Local). In the second condition, we used Euclidean distance for note segmentation and an unquantized note interval representation (Unquantized). In the third condition (Quantized) we used the Euclidean distance for note segmentation and a quantized note interval representation. The quantized condition is equivalent to a previous system we developed [7].

A single trial consists of selecting a condition (Local, Unquantized or Quantized) and performing a training run for one singer, selecting ten of the singers fifteen queries to train on and testing on the remaining five. To speed learning, training was done using a random sample of 250 target songs from the database. The same sample of 250 songs was used for training in each of the three conditions, so that results could be compared fairly. For each trial, the set of parameters with the best training performance was evaluated by finding the MRR of the five testing queries, searching over all 1486 melodies in the database.

We performed three-fold cross validation. Thus, there were three trials per singer for a total of 30 trials per condition. The mean MRR for each of the three conditions is shown in Table 1.

A paired-sample t-test was performed between each pair of conditions and all difference in means were found to be statistically significant ($p < 0.031$ in all cases). A t-test showed all three conditions also performed significantly better than chance ($p < 0.001$ in all cases). This indicates user-specific training does have a positive effect on search results. Further, the use of Mahalanobis distance

Local	Unquantized	Quantized	Chance
0.228(0.14)	0.176(0.14)	0.089(0.09)	0.005

Table 1. Mean (Standard Deviation) MRR over 30 trials in Experiment 1.

for note segmentation and unquantized note interval representation significantly improves performance.

One cause for concern is the variance in the results. In a few trials the learned performance was below chance (< 0.005). In these cases it would make sense to back off to a more generalized set of parameters, learned from a larger population of singers. We explore this idea in Experiment 2.

7.2 Experiment 2

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 the ten singers used in Experiment 1. For these trials we used the Mahalanobis distance for note segmentation and unquantized note intervals. 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. 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 repeated the user-specific training in Experiment 1 with the Local condition. For each trial we determined whether the singer-specific parameters found in a trial had an MRR on the training set (ten songs by one singer) that exceeded by the MRR that would result from using general parameters learned from the other nine singers. If, on the training set, $MRR_u > MRR_g + \epsilon$ we used the user-specific parameters. Else, we used the general parameters. For this experiment, ϵ is 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 (the remaining five songs by that singer). 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 parameter settings is statistically significant ($p = 0.024$).

On 50% of the combined trials the user specific parameters were used and improved performance compared to general training. On 13% of the trials, user-specific parameters were selected, but made performance worse compared to general training. On the remaining 36% of trials, the general data set parameters were used. Figure 4 comaprers the results for conditions.

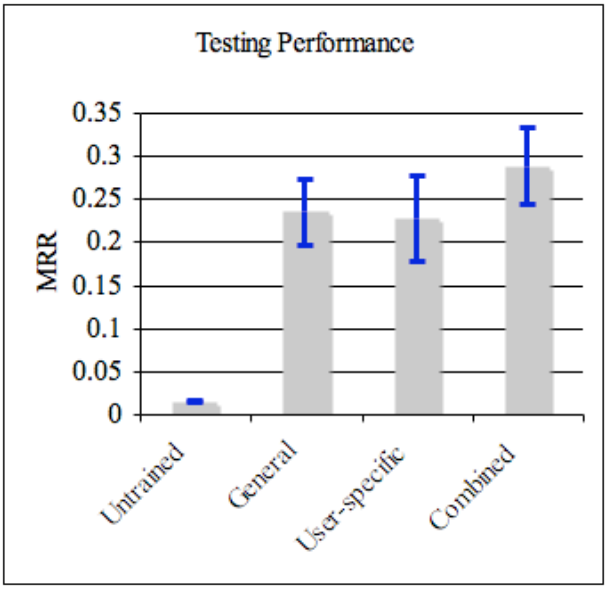

Fig. 4. Testing Performance on Experiment 2.

8 Conclusions

We have described a QBH system that can learn both general and singer-specific error models and note segmentation parameters from labeled user queries. This system can automatically customize parameters to individual users after deployment. Our results show that combining user-specific training with more general training significantly improves mean search performance, resulting in a mean MRR of 0.289 on a database of 1486 melodies. This roughly corresponds to consistently placing the correct target in the top four results. Our results also show both unquantized note intervals and modeling local singer variation for note segmentation significantly improve performance.

In future work we plan to explore how performance varies with respect to the number of training examples provided 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.

References

1. Dannenberg, R., Birmingham, W., Pardo, B., Hu, N., Meek, C., Tzanetakis, G.: A comparative evaluation of search techniques for query-by-humming using the mustart testbed. Jounral of the American Society for Information Sicence and Technology **58**(3) (2007)

2. Kosugi, N., Sakurai, Y., Morimoto, M.: Soundcompass: a practical query-by-humming system; normalization of scalable and shiftable time-series data and effective subsequence generation. In: International Conference on Management of Data, Paris, France (2004) 881–886
3. Pauws, S.: Cubyhum: A fully operation query by humming system. In: ISMIR. (2002)
4. Shifrin, J., Pardo, B., Meek, C., Birmingham, W.: Hmm-based musical query retrieval. In: Joint Conference on Digital Libraries, Portland, Oregon, USA (2002)
5. Unal, E., Narayanan, S., Chew, E.: A statistical approach to retrieval under user-dependent uncertainty in query-by-humming systems. In: Multimedia Information Retrieval Conference, New York, NY (2004)
6. Meek, C., Birmingham, W.: A comprehensive trainable error model for sung music queries. Journal of Artificial Intelligence Research **22** (2004) 57–91
7. Pardo, B., Birmingham, W.P., Shifrin, J.: 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**(4) (2004) 283–300
8. Parker, C., Fern, A., Tadepalli, P.: Gradient boosting for sequence alignment. In: The Twenty-First National Confeerence on Artificial Intelligence. (2006)
9. Pardo, B., Shamma, D.: Teaching a music search engine through play. In: CHI 2007, Computer/Human Interaction. (2007)
10. Pardo, B., Birmingham, W.: Encoding timing information for music query matching. In: International Conference on Music Information Retrieval. (2002)
11. Elder, J.H., Zucker, S.W.: Local scale control for edge detection and blur estimation. Pattern Analysis and Machine Intelligence, IEEE Transactions on **20**(7) (1998) 699–716 0162-8828.
12. Meer, P., Georgescu, B.: Edge detection with embedded confidence. Pattern Analysis and Machine Intelligence, IEEE Transactions on **23**(12) (2001) 1351–1365 0162-8828.
13. Ahmad, M.B., Tae-Sun, C.: Local threshold and boolean function based edge detection. Consumer Electronics, IEEE Transactions on **45**(3) (1999) 674–679 0098-3063.
14. Tzanetakis, G., Cook, F.: A framework for audio analysis based on classification and temporal segmentation. In: EUROMICRO Confrence. Volume 2., Milan (1999) 61–67
15. Shepard, R.: Geometrical approximations to the structure of musical pitch. Psychological Review **89**(4) (1982) 305–309
16. Criani, P.: Temporal codes, timing nets, and music perception. Journal of New Music Research **30**(2) (2001) 107–135 Good/useful representation of pitch similarity here, also some interesting ideas on how to use temporal codes throughout computation.
17. Parker, J.: Genetic algorithms for continuous problems. In: 15th Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence. (2002)
18. Wright, A.: Genetic algorithms for real parameter optimization. In: The First workshop on the Foundations of Genetic Algorithms and Classier Systems. (1990)
19. Jyh-Shing, Jang, R.: Qbsh: A corups for designing qbsh (query by singing/humming) systems (2006) Available at the "QBSH corpus for query by singing/humming" link at the organizer's homepage.
20. Downie, J.S., West, K., Ehmann, A., Vincent, E.: The 2005 music information retrieval evaluation exchange (mirex 2005): Preliminary overview. In: 6th International Conference on Music Information Retrieval. (2005)