

# Teaching a Music Search Engine Through Play

**Bryan Pardo**

EECS, Northwestern University  
Ford Building, Room 3-323, 2133 Sheridan Rd.  
Evanston, IL, 60208, USA  
pardo@northwestern.edu  
+1 (847) 491-7184

**David A. Shamma**

Yahoo! Research Berkeley  
1950 University Ave, Suite 200  
Berkeley, CA 94704  
shamma@yahoo-inc.com  
+1 (510) 704-2419

## ABSTRACT

Systems able to find a song based on a sung, hummed, or whistled melody are called Query-By-Humming (QBH) systems. We propose an approach to improve search performance of a QBH system based on data collected from an online social music game, Karaoke Callout. Users of Karaoke Callout generate training data for the search engine, allowing both ongoing personalization of query processing as well as vetting of database keys. Personalization of search engine user models takes place through using sung examples generated in the course of play to optimize parameters of user models.

## Author Keywords

Karaoke, Query by Humming, Entertainment, Music, Audio

## ACM Classification Keywords

H5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous. H.5.5 [Sound and Music Computing] Methodologies and techniques. Signal analysis, synthesis, and processing.

## INTRODUCTION

When one wishes to find a piece of music on iTunes or at the local library, the usual approach is to enter some textual information about the piece (metadata) such as the composer, performer, or title into a search engine. When one knows the music itself, but not the metadata, standard search engines are not an option. In this case, one might hum or whistle a portion of the piece, providing a query for a search engine based on content (the melody) rather than the metadata. Systems able to find a song based on a sung, hummed, or whistled melody are called Query-By-Humming (QBH) systems.

Popular melodic comparison techniques used in QBH

include string alignment [1], n-grams [2], Markov models [3], dynamic time warping [4] and earth mover distance [5]. These compare melodies transcribed from sung queries to human-encoded symbolic music keys, such as MIDI files.

Deployed QBH systems [6] have no provision for automatically vetting search keys in the database for effectiveness. As melodic databases grow from thousands to millions of entries, hand-vetting of keys will become impractical. Current QBH systems also presume users singing styles will be similar to that of those the system was initially designed for. Individuals vary greatly in singing style and ability, thus, system performance could be greatly improved through effective user personalization after deployment.

We propose a method to improve QBH performance based on data collected from an online social music game, Karaoke Callout, a game where players sing a melody into a mobile phone and a server rates the singing. Playing Karaoke Callout generates training data for the search engine, allowing both ongoing personalization of query processing as well as vetting of database keys.

## DATA COLLECTION WITH ONLINE GAMING

The automated personalization and optimization of our search engine depends on collecting a database of sung examples, paired with correct song titles. What is more, this collection of queries is most useful if the queries are made by the intended users of the system, so that the system has an opportunity to model individual user characteristics.

In previous work, we created VocalSearch [3], a QBH system that allows user personalization of the search engine, based on statistics collected on a corpus of sung examples. VocalSearch uses an error model learned from controlled experiments where a small group of singers were presented with two-note intervals and asked to sing them back. The difference between the example intervals presented to a singer and the intervals transcribed from the sung response was compiled over hundreds of training examples. The result is a statistical model of singer error that greatly improves search results [3].

Our experience with VocalSearch has shown us that typical users do not have the patience to do extensive (on the order of an hour of directed singing) system training. In order to bootstrap the creation of a paired singing-example/target database and encourage user participation, we take a page from recent work in participatory and collaborative tagging. Particular inspirations include Open Mind, the ESP Game and Karaoke Revolution.

The Open Mind Common Sense initiative [7] depends on public participation to generate data for an Artificial Intelligence common-sense-reasoning knowledge base. The ESP Game is a game [8] where two randomly assigned partners are presented with a series of images. The goal for each player is to guess a label their partner would type in for the current image. As the game goes on, the system stores these image tags for later database search. “Karaoke Revolution,” is a popular video game released by Konami for the Sony PlayStation 2, Microsoft Xbox and Nintendo Game Cube, where users sing songs indicated by the computer and are ranked according to how well they match the stored pitch-contour.

These examples have inspired us to cast system training in the form of a prototype interactive, client-server karaoke game: *Karaoke Callout* [9]. This game closes the loop between system use and system improvement by providing correct song labels for sung examples, so that we can automatically vet and update a musical search engine.

**Karaoke Callout**

We have created an initial proof-of-concept for Karaoke Callout, written in Python that works on a Symbian S60 cellphone. In the game, the user selects a song and sings it into the phone. The audio is sent to our current VocalSearch music search engine, which rates the quality of the singing by measuring how closely it resembles hand-coded melodic key in the server database, sending a score back to the user. The user may then challenge anyone in the phone’s contact list. An SMS text challenge is sent to the challenged person’s cell phone. The challenged person sings the song, attempting to better the performance of the challenger. This challenge may then be repeated, with either party selecting a new song with which to “call out” the other party. Over the course of an interaction, numerous examples of each party’s singing are created and stored. Figure 1 shows a series of screen shots from the cell phone over the course of one round of play.

Karaoke Callout is not only a data-collection tactic, but a social and collaborative music gaming system in its own right. Karaoke callout rethinks the asynchronous interaction of text messaging to allow a new kind of musical interaction at a distance in both space and time. Our approach is distinct from prior efforts to utilize the internet for music such as Elaine Chew’s work [10] and the SoundWire project [11] in that the focus will not be on

telepresence and live interaction of expert musicians. Rather, the goal is community enhancement through interactive, multi-player music gaming. This allows us to focus on building intuitive and innovative music applications, rather than on issues of network latency and high-tech Net 2.0 installations.

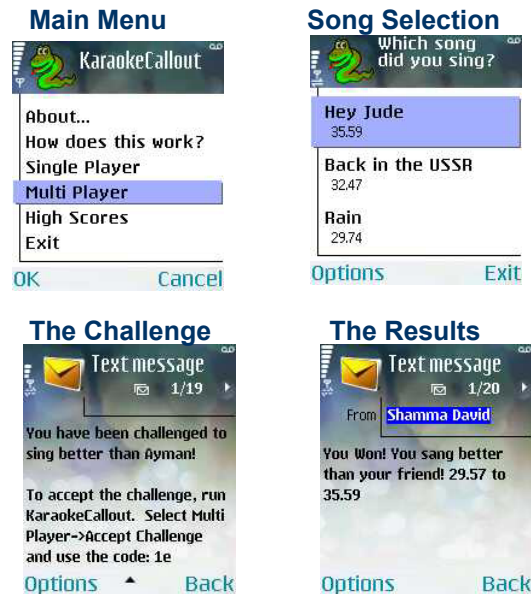


Figure 1. Karaoke Callout interaction

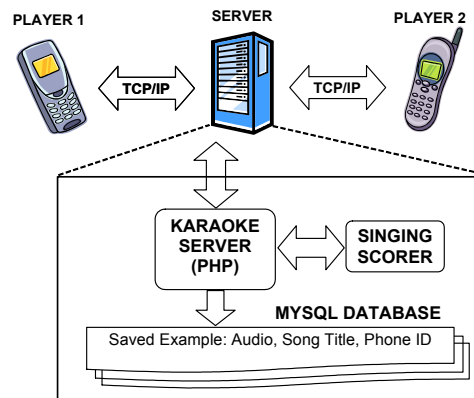


Figure 2. Karaoke Callout System Overview

The game server (see Figure 2) is divided into three main components. The first of these is the Karaoke Server (written in PHP), which handles communication with the clients, queries the Singing Scorer (our VocalSearch search engine) and stores sung examples in the database. The final component is a MySQL database of audio queries, scores, and challenges. The Singing Scorer is modular and separate from the Karaoke Server, allowing each component to be updated independently. This is key for implementing automated system personalization and learning, as the singing scorer is the search engine that we wish to optimize.

## A QUERY DATABASE FOR SYSTEM OPTIMIZATION

When a user queries for a particular song, such as “Lola”, we consider a search engine successful if the correct target (Lola, in this case) is returned as one of the top few answers. The closer the target gets to number one, the better the system performance. When a single search fails, it may be difficult to tell exactly why. The query may be poorly formed (singing “Hey Jude” when searching for “Lola”), the search method may be ineffective for a particular user (the user model needs optimization), or the individual search key may not correspond well with what a typical person would sing (storing only the verse when people sing only the chorus). Maintaining a database of past queries and their associated targets makes it possible to distinguish between these cases and react appropriately.


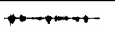
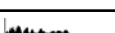
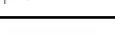

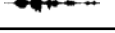

| User | Query Audio   | Target Title  | Target Rank |
|------|---|---------------|-------------|
| 1    |    | Hey Jude      | 190         |
| 1    |    | Que Sera Sera | 39          |
| 1    |    | Lola          | 21          |
| 2    |    | Hey Jude      | 233         |
| 2    |    | Lola          | 1           |
| 3    |   | Hey Jude      | 59          |
| 3    |  | Que Sera Sera | 1           |

Table 1. Stored queries to a QBH system

Consider Table 1. Each row in the table corresponds to a single query made to a search engine. Or, in the case of Karaoke Callout, a sung example for the game. Here, “Query Audio” is the recorded singing, “Target Title” is the title of the correct target in the database and “Target Rank” is the rank of the correct target in the results returned by the system. In this example, every query by user 1 failed to place in the top ten. This is an indication that the search engine is not optimized properly for this user. Note also that every query for “Hey Jude” failed to place within the top fifty, regardless of user. This indicates a mismatch between the target and the kinds of query example users provide. This is in direct contrast to both “Que Sera Sera” and “Lola,” each of which has one query whose correct target was ranked first.

A database containing sung examples, keyed to correct song titles lets us automatically vet our search keys by using them as example queries. Those targets with consistently below-average search results can then be tagged for search key updating. Such a database also allows for principled, automatic improvement of our similarity measures, personalizing the system to individual users. We

incorporate such a learner into the system so that singer models can be learned for individuals that have sung sufficient numbers of examples. For new users, a generic model will be used until enough user-specific information can be collected to perform individual modeling.

## SYSTEM OPTIMIZATION AND PERSONALIZATION

We implement a straightforward method for automatically updating a low-performing search key by generating the new key from one of the past queries. The effectiveness of this new key can be measured by rerunning the saved queries against this new key. This can be repeated using a key based on each query (or even on the union of all queries) and the best new key may then replace or augment the original search key. This allows automatic, constant updating and improvement of the database without need for expert intervention.

A primary measure our system optimizes is *mean reciprocal right rank* (MRRR), shown in Equation 1. The *right rank* of a query is the rank of the correct song for the query. The mean right rank for a trial is the average right rank for all queries in the set.

$$MRRR = \frac{\sum_{n=1}^{numberOfQueries} \frac{1}{rightRank_n}}{numberOfQueries} \quad (1)$$

We use MRRR because it gives more useful feedback than the simple mean right rank. Consider the following example. System A returns right ranks of 1, 1, 199, and 199 for four queries. System B returns 103, 102, 98, and 97. We prefer a system that ranks the correct target 1<sup>st</sup> half of the time to one that ranks it around 100<sup>th</sup> every time. Mean right rank returns a value of 100 for both systems. MRRR returns 0.5 for system A and 0.01 for system B.

When vetting targets, one need only measure MRRR for each target in the database. When the MRRR of a particular song falls below a given value, it becomes a candidate for replacement, as described above. We automate personalization of our note segmentation and singer error models in a similar way.

Before a melodic comparison takes place, our transcriber estimates the fundamental frequency of the singing every 20 milliseconds. The note segmenter then divides this series of estimates into notes. Equation 2 shows our note segmentation weighting function. Here,  $a_i$ ,  $f_i$ , and  $h_i$  represent, respectively, the amplitude, fundamental frequency and harmonicity estimates for the  $i$ th frame.

$$d = \alpha |a_i - a_{i-1}| + \beta |f_i - f_{i-1}| + \chi |h_i - h_{i-1}| \quad (2)$$

When the weighted linear combination exceeds a threshold, a new note is begun. Our current system uses a simple hill-climber with random restarts to estimate the ideal for this function. It is a simple matter to optimize for a singer by simply tracking the MRRR of the stored singing examples for that particular singer, when these examples are treated as queries by the search engine.

Our singer pitch-interval error model is treated in a similar fashion. Error is presumed to be symmetrically distributed around the correct pitch interval in a Gaussian distribution. Hill climbing is again used to find the optimal standard deviation for the distribution. As with note segmentation, this is done by measuring MRRR on the database of singing examples gathered from Karaoke Callout.

### CONCLUSIONS

Our formal study and system evaluation awaits IRB approval/human subject protocol approval. Currently, data has been collected only during application testing. Our small number of test singers shows an increase in their overall system performance after training. Interestingly, performance still improves (though not as much) with system training disabled. One particular tester's scores increased significantly over 2 weeks. He complemented the system's ability to 'learn' his singing behavior so well—unaware training was disabled. We are constructing a formal study, based on this anecdotal data, to investigate how people may adapt to a query-by-humming system as well as how individually adaptive a query-by-humming systems need to be.

While the current client works on any web-enabled phone using Python for S60 on the Symbian S60 Series 2 operating system, such phones are a very small subset of the installed base of cell phones. To increase the base of potential users we will also create a java-based web-client for Karaoke Callout so that any computer with an internet connection can be used to participate in a callout. With respect to cell phones, we plan to make a new client for Java2 Mobile Edition (J2ME) enabled devices to reach a much wider audience. While this will pose some challenges, as audio APIs are not standardized across platforms we minimize complications by doing all audio processing (other than simple recording) on the server.

We also plan to add location information into Karaoke Callout using the logical location specified by the cell phone's active tower connection, a technique that has been used for social photo spaces, such as Yahoo's ZoneTag project ([zonetag.research.yahoo.com/](http://zonetag.research.yahoo.com/)). We hope to motivate players by creating a karaoke challenge based on location. A player could thus challenge anyone in their local town who has the Karaoke Callout client installed and ranking could be kept on a regional basis. A player in Berkeley, California, for example, will be able to see the high scores and most popular songs in Berkeley, as well as have the ability to challenge those songs and take

ownership of the karaoke space within that neighborhood. This information could also prove useful for localized music search optimization of our search engine.

### ACKNOWLEDGMENTS

We thank David Little and David Raffensberger for assistance in this project.

### REFERENCES

1. Pauws, S. CubyHum: A Fully Operational Query by Humming System. in *Proceedings of ISMIR 2002*. 2002. Paris, France.
2. Doraisamy, S. and S. Ruger. A Comparative and Fault-tolerance Study of the Use of N-grams with Polyphonic Music. in *Proceedings of ISMIR 2002*. 2002. Paris, France.
3. Pardo, B., 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*, 2004. 55(4): p. 283-300.
4. Adams, N.H., Bartsch, M.A., Shifrin, J. B., Wakefield, G. H. Time series alignment for music information retrieval. in *ISMIR 2004, 5th International Conference on Music Information Retrieval*. 2004. Barcelona, Spain: University of Michigan.
5. Typke, R., P. Giannopoulos, R.C. Veltkamp, F. Wiering, and R. van Oostrum. Using transportation distances for measuring melodic similarity. in *ISMIR 2003, 4th International Conference on Music Information Retrieval*. 2003. Baltimore, MD.
6. Typke, R., F. Wiering, and R.C. Veltkamp. A Survey of Music Information Retrieval Systems. In *ISMIR 2005: 6th International Conference on Music Information Retrieval*. 2005. London, England.
7. Singh, P., The public acquisition of commonsense knowledge, in *Proceedings of AAAI Spring Symposium on Acquiring (and Using) Linguistic (and World) Knowledge for Information Access*. 2002, Palo Alto, CA.
8. von Ahn, L. and L. Dabbish. Labeling Images with a Computer Game. in *CHI 2004*. 2004. Vienna, Austria.
9. Shamma, D. and B. Pardo. Karaoke Callout: using social and collaborative cell phone networking for new entertainment modalities and data collection, in *Proceedings of ACM Multimedia Workshop on Audio and Music Computing for Multimedia (AMCMM 2006)*. 2006. Santa Barbara, CA, USA.
10. Chew, E., A.A. Sawchuk, and R. Zimmermann. Distributed Immersive Performance: Ameliorating the Psychophysical Effects of Network Latency. in *Proceedings of the Corporation of Education Networks in California (CENIC)*. 2005. Marina del Rey, CA.
11. Chafe, C., QoS Evaluation with SoundWIRE, N.S.F Grant, 1999, OCI Office of CyberInfrastructure.