SPEEDING MELODY SEARCH WITH VANTAGE POINT TREES

Michael Skalak, Jinyu Han, Bryan Pardo
Electrical Engineering and Computer Science
Ford Engineering Design Center, Room 3-323
Northwestern University
2133 Sheridan Road, Evanston, IL, USA 60208
847.491.7184
mskalak13@gmail.com, jinyuhan@gmail.com, pardo@northwestern.edu

ABSTRACT

Melodic search engines let people find music in online collections by specifying the desired melody. Comparing the query melody to every item in a large database is prohibitively slow. If melodies can be placed in a metric space, search can be sped by comparing the query to a limited number of vantage melodies, rather than the entire database. We describe a simple melody metric that is customizable using a small number of example queries. This metric allows use of a generalized vantage point tree to organize the database. We show on a standard melodic database that the general vantage tree approach achieves superior search results for query-by-humming compared to an existing vantage point tree method. We then show this method can be used as a preprocessor to speed search for non-metric melodic comparison.

1. INTRODUCTION

Music is a popular category of online multimedia. Example collections include the millions of recordings in iTunes, emusic, and amazon.com. New melodic search engines let one search a music collection by specifying the desired melody. Typically, the query is entered by singing, playing a virtual piano keyboard or entering a symbolic representation, such as notes on a music staff. For an overview of recent approaches to melodic search, see [1, 2] or the results of the MIREX competitions [3].

Currently deployed systems compare the query melody to each melodic search key in the database. While this works for small collections, direct comparison to every database element becomes prohibitively slow as the size of the collection increases. Placing database melodies in a metric space lets one leverage a large body of research on efficiently finding objects [4].

Given a metric for melodies, we can pre-compute the distance of each melody in the database to a small set of melodies taken from the database, called vantage points. By comparing the query's distance to the vantage points with those of the database melodies, many melodies can be removed from consideration without need for direct comparison to the query, speeding search.

Recently, several authors [5-7] have organized melodic databases with vantage points and a metric. Typke et al [6] made an excellent first step, encoding melodies as a piecewise constant functions on the pitches of the chromatic scale and then applying a variant of Earth Mover Distance (EMD) [8]. EMD forms a pseudometric (non-identical elements may have a distance of 0). Their work did not, however, use a true metric, nor did it focus on learning to optimize the metric from real query data. They also did not explore the potential benefits of organizing vantage points in a structure.

Parker et al. [5] approximated an edit-distance-based metric for melodies based on a data set. They used a recursively organized structure, called a *vantage point tree* [4, 9], to prune the search space. Their system showed good results, but they did not learn a true metric, potentially compromising effectiveness. As with Typke et al., their pitch quanta are the notes on the chromatic scale. Such quantization can introduce error when used with pitch-tracker output from audio recordings (like sung queries) [10]. Also, they only tested a single tree structure, leaving open the possibility that other tree structures could improve performance.

This paper describes an un-quantized melodic encoding (Section 2) and a simple metric for melodic comparison (Section 3). We use this metric to organize a melodic database with a more general vantage point tree architecture than previous work has used (Section 4). The simplicity of the metric lets us tune its parameters (Section 6) from a small set of sung queries (Section 7). We explore the space of vantage point trees to find architectures that greatly improve search speed while sacrificing little in terms of search quality (Section 8). We then show a vantage point tree can be an effective preprocessor to speed search for a non-metric melodic comparison approach (Sections 5 and 9).

2. MELODIC ENCODING

In a typical melodic search system, the query is provided to the computer quantized to some musical alphabet (e.g. the note staff entry at http://musipedia.org) or as a sung example (http://midomi.com). In the case of a sung example, the melody is typically transcribed into a time-frequency representation where the fundamental frequency and amplitude of the audio is estimated at short fixed intervals (on the order of 10 milliseconds).

Our system accepts both music-staff entry and sung queries, so we use a representation scheme that is useful for both these cases. We encode all queries and all melodies in the database as sequences (strings) of note intervals. Each *note interval* is represented by a pair of values: the pitch interval (PI) between adjacent notes (measured in units of musical half-steps) and the log of the ratio between the length of a note and the length of the following note (LIR) [11] (where note lengths are inter-onset-intervals). We use note intervals because they are transposition invariant (melodies that differ only in key appear the same) and tempo invariant (melodies that differ only in tempo appear the same).

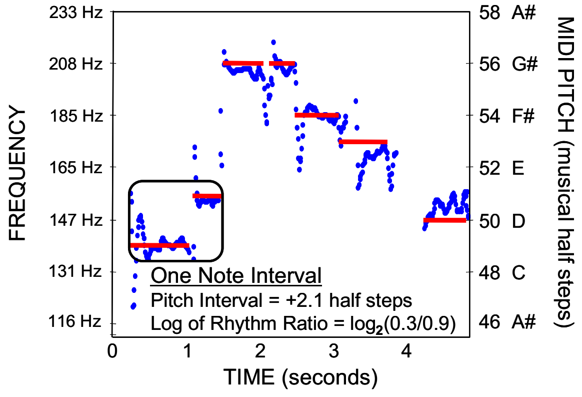

Figure 1. Encoding of a sung query

Figure 1 shows a transcription of a sung query by our system. Dots show the initial transcription. Horizontal lines show notes derived from the transcription. The rounded rectangle surrounding two notes indicates a single note interval derived from those two notes.

3. THE METRIC

A metric is a function $d(x,y)$ that defines the distance between elements x and y of a set. A metric must satisfy the following conditions: $d(x,y) \geq 0$ (non negativity); $d(x,y) = 0$ iff $x = y$; $d(x,y) = d(y,x)$ (symmetry); and $d(x,y) + d(y,z) \geq d(x,z)$ (triangle inequality).

While there are any number of functions that satisfy the requirements of a metric, most are unsuitable for meaningful melodic comparison. Since our focus is on speeding melodic search, a good metric for melodies must be quick to calculate and one where $d(X,Y)$ is small when a person would say melodies X and Y are similar and $d(X,Y)$ is large when a person would call X and Y dissimilar. We now describe a simple metric that has proven effective for melodic comparison.

We represent a melody X as a string of note intervals. We denote note intervals as lower case letters. Equation 1 defines a simple metric between note intervals x and y, with pitch intervals x_p and y_p and LIRs x_l and y_l.

$$d(x,y) = a\left|x_l - y_l\right| + b\left|x_p - y_p\right| \tag{1}$$

Here, a and b are non-negative values chosen to optimize performance on a set of example queries for a given database. This simple approach, when paired with the differential melodic encoding described in Section 2 (this encoding is crucial to the use of such a simple note metric), has been shown to produce comparable search performance to more complex distance measures, without the need to optimize large numbers of parameters [2].

The proof that this is a metric is straightforward: the absolute difference between two values is a metric. So is the weighted sum of two metrics. Thus, our distance measure is a metric on note intervals. Note this metric does not force quantization of the note values to a finite alphabet. Such quantization has been shown to introduce errors in melodic search using sung queries [10].

The distance between the query and each database element determines its ranking in search results. Of course, when searching in a melodic database, one is not comparing individual note intervals, but full melodies. To compare melodic strings, we use global edit distance. The edit distance between two strings is the cost of the least expensive way of transforming one string into the other. Here, transformation cost depends on a comparison function for the individual string elements. We have a fixed insertion/deletion cost of one, effectively forcing the other parameters to be in these units. If the comparison function for string elements is a metric (like Equation 1) then edit distance is also a metric. For a more in depth description and proof, see [12, 13].

4. VANTAGE POINT TREES

We now give an overview of the vantage point tree algorithm [9] for speeding search. We organize a database by choosing v elements at random from the database. These elements are the vantage points. Each element in the database has its distance measured to each vantage point using a metric. Around each vantage point, we partition the space into r concentric rings. We choose the size of these rings so that each contains nearly equal numbers of elements.

Given v vantage points with r rings per vantage point, this divides the metric space into r^v regions of intersection between rings, called branches. Each branch can be uniquely identified by a *v*-tuple where the *i*th element specifies the ring for the *i*th vantage point. This tuple is the branchID. All database elements in the same branch are assigned the same branchID. BranchID values for database elements are computed prior to search.

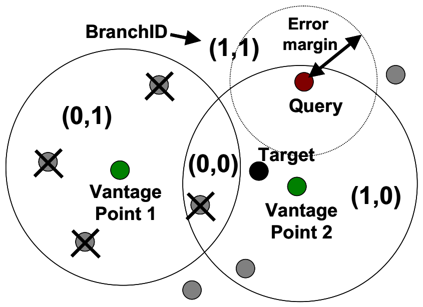

Figure 2. One level of a vantage point tree

Figure 2 shows one level of a vantage point tree with two vantage points and two rings per vantage points. Points in the graph indicate database elements (i.e. melodies). Each vantage point in this example has only two rings: a close inner region and a far outer region. This splits the space into four branches. In Figure 2, each branch ID is a binary duple.

To search the database, we find all branches that intersect the spherical region within an empirically-determined error margin around the query. This set of branches contains the only points we need to consider when searching the database. In Figure 2, this eliminates all melodies in branch (0,1) and (0,0), leaving the melodies in branches (1,0) and (1,1) as possible matches.

The error margin around the query represents our uncertainty about how well the metric corresponds to the choices a human would make in selecting the most similar melody. We are guaranteed to find all targets that are within the error margin of the query. However, the smaller the error margin, the fewer branches fall within it, speeding search. The larger the error margin, the less likely we are to accidentally remove the correct target from consideration.

Even a single-layer tree can eliminate a large number of database elements from consideration. The true power of this approach, however, lies in creating levels. Within any branch at level *l*, one can partition the space again by selecting vantage points within that branch and creating a set of branches at level *l*+1. This recursive structure is called a vantage point tree. We apply the algorithm at each level, until a maximum depth is reached or the leaves are too small to split further. At this depth, the remaining database elements are directly compared to the query to find the best match. The cost of searching the tree is negligible other than the comparisons to the vantage points; at worst, it requires a number of hash table lookups equal to the non-eliminated branches at each level. Given a poor metric (no branches are eliminated at any level) worst case performance is O(*n*). Typical performance is O($\log_B(n)$). Log base *B* depends on the metric and tree structure. Different structures can have similar search quality, while varying in speed by an order of magnitude. This is shown in Section 8.

The vantage point tree used by Parker et al [5] is a special case with one vantage point per level (*v*=1) and two rings per vantage point (*r*=2). Similarly, the structure used by Typke et al [6, 7] can be realized with setting of *v* to the desired number of vantage points, *r* equal to the number of elements in the database, and only a single level (maximal tree depth *d*=1). Setting *v*=1, *r*=1 and *d*= 1 results in only one ring and all melodies must be directly compared to the query. This is linear search.

5. VANTAGE POINT TREE AS PREPROCESSOR

The metric in Section 3 is based on edit distance. Recent MIREX competitions show that recursive alignment (RA) generates a higher MRR than does the string-edit approach we can prove is a metric [14]. Unfortunately, RA is not a metric. Thus, we cannot directly apply vantage point trees to recursive alignment. We can, however, still use a vantage point tree preprocessor to speed search for a non-metric.

Let distance measure *M*(*a,b*) be a metric. Let *H*(*a,b*) be a slow non-metric we assume is ground-truth. If *M* and *H* agree within error bound *e*, we can build a vantage point tree with *M* and search with an error radius based on *e*. This precludes false negatives due to differences between *M* and *H*. The remaining database elements can be sorted using *H*. This lets us speed search by preprocessing with a vantage point tree built with *M*. We can progressively shrink the error bound to further speed search at the cost of increasing numbers of false negatives.

If we could find a vantage point tree able to put most queries in the same branches as their targets, it could be a useful preprocessor for any melodic search engine. Because each branch contains only a small portion of the database, the preprocessing could greatly speed up search, while still giving a high probability that the targets are within the search scope. Section 9 describes an experiment to find a good vantage point tree for this purpose.

6. DATA SET

Although there is nothing inherent in our approach that requires sung queries, we focus our experiments on the query-by-humming melodic search case, where queries are audio recordings of people humming or signing melodies. Our query set was drawn from the QBSH corpus [15] used in the 2006 MIREX comparison of query-by-humming melodic search systems [3]. We used 15 singers, each singing the same 15 songs from this dataset for a total of 225 queries. Our melody database contained 2348 folk melodies from the Essen database (http://www.esac-data.org/) plus the 15 target melodies corresponding to the sung melodies in the query set, for a total of 2363 melodies. This database was chosen to emulate the database used in the 2006 MIREX competition.

The median melody length was 40 notes. Melodies were split into 10 note-interval subsequences with 3 note-intervals between the start of each subsequence and the search rank of the best subsequence was used as the rank of the melody. Melodies were broken into 11 subsequences, on average, resulting in 34,000 total database elements. While this is small compared to the millions of songs available in iTunes, the dataset used is a standard one used in the music search community. Copyright issues and the need to hand-vet melodies have limited the size of data sets used by this community.

7. TUNING THE METRIC

The metric from Section 3 may be tuned to favor pitch or rhythm similarity. Given the small number of parameters to be tuned, the model may be tuned using a relatively small set of examples. We learned parameters for a note segmenting preprocessor [10] and the note interval metric from Equation 1 using a simple genetic algorithm (GA) [16]. For tuning the metric, we selected five singers and used five of their songs for a total of 25 queries in our training set. Our testing set consisted of all 15 songs from each of the remaining 10 singers (150 queries). The GA used fitness proportional reproduction and single-point crossover. Each individual in the population was one set of segmentation and metric parameter values. To test the fitness of a set of parameter values, we ran a search engine using those values on the database from Section 6. Fitness for the GA was determined by MRR (see Section 8) on the training set and final results were validated on the testing set. Once good values for the metric were found, these values were used consistently in the experiments reported in this paper.

8. COMPARING TREES

We are interested in effect of tree architecture on the performance of a vantage point tree. To explore the space of the parameters for the vantage point tree, we generated 1736 random four-tuples (*v,r,d,e*). These represent the number of vantage points per level (*v*), rings per vantage point (*r*), maximal tree depth (*d*), and radius of the error margin around the query (*e*). Here *d* ranged from 1 to 20 levels, *v* ranged from 1 to 10, *r* ranged from 2 to 50, and *e* ranged from 0 to 10.

Trees were evaluated by the performance of a search-engine in finding correct targets for queries using the database from section 6. Given a vantage point tree, each query was processed. Once the final leaf branch was reached, all database melodies in the leaf were ranked for similarity to the query using the metric from Section 3. If the correct target was not in the same leaf as the query, we treated it as having been ranked 100th.

Call *c* the rank of the correct target for a query. Here, *c* = 1 indicates the correct target was ranked first. The *mean rank* for a trial is the average value for *c* over all queries in the trial. This measure can be sensitive to poorly ranking outliers. The *mean reciprocal rank* (MRR) is less sensitive to outliers. The reciprocal rank is 1/*c*. MRR (see Equation 2) is the mean of the reciprocal rank over all queries in a trial. Here, *Q* is the number of queries and *N* is the size of the database.

$$1 \geq MRR = \frac{\sum_{q=1}^{Q} \frac{1}{c_q}}{Q} \geq \frac{1}{N} \tag{2}$$

If the system always ranks a correct answer first, the MRR will be 1. If the system gives random similarity values to targets, the MRR will be roughly log(N)/N. Chance MRR performance on our database is 0.03.

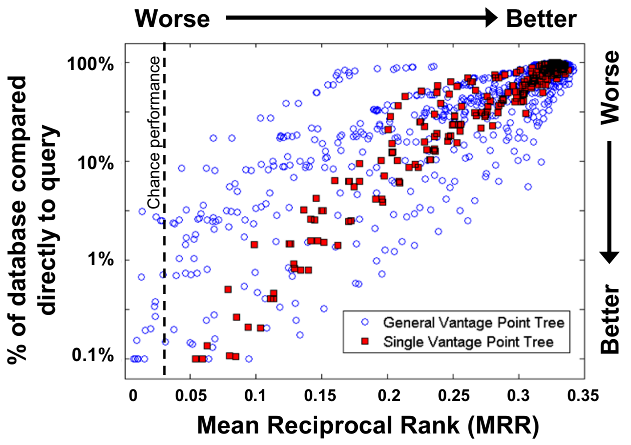

Figure 3. Performance of vantage point trees plus the metric as a stand-alone melodic search method.

Results for the 1736 vantage point trees are shown in Figure 3. Each point represents the performance of the search engine on the dataset from Section 6, given vantage tree parameter settings (*v,r,d,e*). Results show mean values for the 200 query melodies not used in training the metric. Parameter settings conforming to the simple vantage point tree method described in Parker et al [5] (Single Vantage Point Trees) are shown as solid squares. Trees with more than two rings per vantage point and/or more than one vantage point per level are shown as open circles (General Vantage Point Trees). The vertical dimension shows the proportion of the database compared directly to the query. Thus, a value of 1% means that 99% of the database was removed through use of the vantage point tree, prior to the direct comparison of the final 1% to the query.

As Figure 3 shows, for every Single Vantage Point Tree, there is a General Vantage Point Tree that eliminates the same proportion of the database, while achieving a higher MRR. This indicates that using more complex vantage point tree structures can significantly improve search performance.

Figure 3 shows a general vantage point tree allows a string-alignment based method to achieve an MRR of 0.33 by comparing only 10% of the melodies in the database to the query melody. Full search of the database using the same method achieves an MRR of 0.346. This is an advance for edit-distance based search. Other search methods may also benefit from the metric and vantage point tree by using it as a front end. We now describe an experiment that explores this possibility.

9. FINDING A GOOD FRONT-END TREE

When a query is processed with a vantage point tree, we find which branches intersect the spherical region described by the error margin around the query. These branches are then divided into smaller branches at the next level. The tree is traversed until some depth limit is reached. At this point, we say the target song is still in our vantage point tree if at least one subsequence of the target melody is in one of the non-eliminated branches at the final level. The melodies in these branches may then be compared to the query by any desired search method.

By varying the value of error margin around the query (see Figure 2), we can change the number of the branches in the tree under consideration. Generally speaking, the larger the error margin is, the fewer branches will be eliminated from consideration. We wish to find an error margin that eliminates the largest portion of the database without removing the target melody.

To find a good tree architecture as a front-end for melodic search, we randomly generated 80 vantage point tree architectures by setting the number of vantage points and number of rings per point (v, r). For each pair of parameter settings, we then organized the database from Section 6 with a vantage point tree, recursively creating branches until reaching a level where the branches contain less than $2(v+r)$ melodies.

We then chose 150 queries and randomly selected 100 queries (the selecting set) from these 150 queries. We queried each tree with each query in the selecting set and counted the number times the correct target remained in the database after applying the vantage point tree. Call this t. We then calculated the percentage of database left after applying the tree, p. For each tree, we took the ratio t/p as a measure of the effectiveness of the tree. We then ranked the trees by this ratio and took the best one as the good vantage tree. This "good vantage tree" architecture was tested using the remaining 50 queries. We repeated this selecting and testing process three times using 3-fold cross validation. The best tree architecture from each of the three trials was *Trial1*: v=4 ,r=3, *Trial2:* v=3, r=5, and *Trial3:* v=3, r=5.

Figure 4 shows average performance over the three trials. Open circles with the dashed line show performance on the testing set. Solid circles with the solid line show performance on the selecting (training) set. The numeral by each point indicates the error margin around the query (see Section 4). The diagonal line shows the effect of shrinking the database by randomly removing a given percentage of the elements.

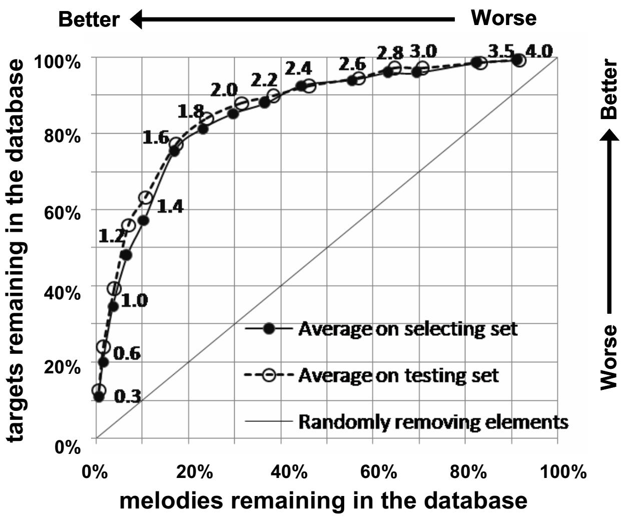

Figure 4. Performance of vantage point trees as a front-end for a melodic search system.

One may select the appropriate trade-off between search quality and search speed by choosing the appropriate error margin. For example, an error margin of 2.4 leaves 90% of the targets in the database, while eliminating 50% of the other database elements. An error margin of 1.7 retains 80% of the correct targets, but only 20% of the database, speeding search by a factor of five.

After narrowing the search scope by the vantage point tree preprocessor, one could use any melodic comparison approach for the remaining elements. This could make some time-consuming comparison methods practical in current computing-limited systems. If the errors made by the vantage point tree are uncorrelated with errors the subsequent search method makes, we could even improve MRR, since the tree could eliminate database elements that might confuse the other search method.

10. CONCLUSIONS

This paper describes a way to speed melodic database search. We apply a simple parameterized approach to building a metric for melodies. This simplicity lets us learn good metric parameter values from a small set of queries. We use this metric to organize a database to speed search, employing a more general vantage-tree approach than existing work has used. This results in significantly improved search times while sacrificing very little search quality. Previous methods used to index a melodic database with vantage point trees are special cases in this framework. Any melody metric can be used with this approach to database organization. For example, one could choose one metric for skilled musicians and another for the general public.

One can also use a vantage point tree preprocessor to speed search for a non-metric. By choosing the correct parameters for the vantage point tree, one can balance search speed against the likelihood of eliminating the correct target from the database.

Optimal tree architecture for a particular application depends on the metric and the content of the database. In future work, we will explore the relationships between tree architecture, metric and dataset to improve our ability to select the right tree architecture.

11. ACKNOWLEDGEMENTS

This work was funded in part by National Science Foundation Grant number IIS-0643752.

12. REFERENCES

[1] 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.

[2] Dannenberg, R., 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: p. 687 - 701.

[3] Downie, J.S., K. West, A. Ehmann, and E. Vincent. The 2005 music information retrieval evaluation exchange (MIREX 2005): Preliminary overview. in *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*. 2005. London, England.

[4] Chavez, E., G. Navarro, and J.L. Marroquin, Searching in Metric Spaces. *ACM Computing Surveys*, 2001. 33(3): p. 273-321.

[5] Parker, C., A. Fern., and P. Tadepalli. Learning for Efficient Retrieval of Sturctured Data with Noisy Queries. in *Proceedings of the 24th International Conference on Machine Learning (ICML)*. 2007. Corvalis, Oregon.

[6] Typke, R., R.C. Veltkamp, and F. Wiering. Searching Notated Polyphonic Music Using Transportation Distances. in *Proceedings of ACM Multimedia 2004*. 2004. New York, NY, USA.

[7] Vleugels, J. and R.C. Veltkamp. Efficient Image Retrieval through Vantage Objects. in *Proceedings of the Third International Conference on Visual Information and Information Systems*. 1999.

[8] 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. Balitmore, MD.

[9] Yianilos, P.N. Data Structures and Algorithms for Nearest-neighbor Search in General Metric Spaces. in *Proceedings of the Fourth annual ACM-SIAM Symposium on Discrete Algorithms*. 1993.

[10] D. Little, D.R., B. Pardo, User specific training of a music search engine, in *Machine Learning and Multimodal Interaction: Fourth International Workshop, MLMI 2007*, Lecture Notes in Computer Science. 2007, Springer: Brno, CZ.

[11] Pardo, B. and W.P. Birmingham. Encoding Timing Information for Musical Query Matching. in *ISMIR 2002, 3rd International Conference on Music Information Retrieval*. 2002. Paris, France.

[12] Levenshtein, V.I., Binary Codes Capable of Correcting Deletions Insertions and Reversals. *Soviet Physics Doklady*, 1966. 10(8): p. 707-710.

[13] Wagner, R. and M. Fischer, The string-to-string correction problem. *Journal of the ACM*, 1974. 21(1): p. 168-173.

[14] Wu, X., Li, M., Liu, J. , Yang, J. , and Yan, Y. A Top-down Approach to Melody Match in Pitch Contour for Query by Humming. in *International Symposium on Chinese Spoken Language Processing*. 2006.

[15] Jyh-Shing and R. Jang, QBSH: A corpus for designing QBSH (query by singing/humming) systems. 2006, Available at the http://www.cs.nthu.edu.tw/˜jang.

[16] 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.