A Two-stage Query by Singing/Humming System on GPU

Wei-Tsa Kao, Chung-Che Wang, Kaichun K. Chang, Jyh-Shing Roger Jang, and Wenshan Liou
ISA, National Tsing Hua University, Hsinchu, Taiwan, R.O.C.
Department of CS, National Tsing Hua University, Hsinchu, Taiwan, R.O.C.
King's College London
Department of CSIE, National Taiwan University, Taipei, Taiwan, R.O.C.
Smart Network System Institute, III, Taipei, Taiwan, R.O.C.
E-mail: {jeffery.kao, chungche.wang}@mirlab.org, Ken.chang@kcl.ac.uk, jang@csie.ntu.edu.tw, wsliou@iii.org.tw

***Abstract*—this paper proposes the use of GPU (graphic processing unit) to implementing a two-stage comparison method for a QBSH (query by singing/humming) system. The system can take a user's singing or humming and retrieve the top-10 most likely candidates from a database of 8431 songs. In order to speed up the comparison, we apply linear scaling in the first stage to select candidate songs from the database. These candidate songs are then re-ranked by dynamic time warping to achieve better recognition accuracy in the second stage. With the optimum setting, we can achieve a speedup factor of 7 (compared to dynamic time warping on GPU) and an accuracy of 77.65%.**

I. INTRODUCTION

Query by singing/humming (QBSH) is an intuitive method for music retrieval. When a user sings or hums a tune to a microphone, the audio signal is sent to the server for pitch tracking, and the QBSH engine can compare the query pitch with database songs to retrieve the most similar ones. Our QBSH system, called MIRACLE [1], adopts various comparison methods to improve the accuracy. In an earlier version, MIRACLE applies linear scaling (LS) or dynamic time warping (DTW) on a set of clustered machines. Inspired by a number of computation intensive applications over GPU (graphic processing unit), we have implemented a GPU-based MIRACLE system in 2012 [2] with LS for comparison.

With the availability of GPUs, some real-time applications become possible. Poli et al. [3] applied dynamic time warping (DTW) on voice password identification system on GPUs, where four-digit passwords were used in their experiments. Li et al. [4] evaluated the probability of hidden Markov models on GPUs, where forward probability were calculated and summed up in parallel.

There are still some similar studies of accelerating QBSH on GPUs. One of them is purposed by Ferrao et al. [5], which used local transposition alignment to retrieve intended songs on GPU. However, the comparison only starts from the beginning of a song. Our previous work [2] applied LS to QBSH, where the comparison could starts from anywhere in a song. Kuo [6] extends the work of [2] by using DTW for QBSH.

Some other studies attempt to combine different methods of QBSH or perform multiple-stage QBSH. Lin and Fang ([7], [8]) have proposed the two-stage linear scaling on embedded system constrained by limited memory and computing power. Ho et al. [9] has introduced highest rank, Borda Count, and logistic regression methods for decision combination. Zou [10] has combined linear scaling and dynamic time warping in parallel and serial combination by using Borda Count with better performance. We also adopt this method for score combination and reranking.

This paper proposes a two-stage comparison method, which combines LS and DTW on GPU for achieving better accuracy. At stage 1, LS is used to choose candidate songs. DTW is then applied to re-rank the candidates in stage 2. We also investigate how to optimally combine scores by LS and DTW instead of simply re-ranking them.

The rest of this paper is organized as follows. The fundamentals of LS and DTW are described in section 2. The proposed two-stage method is introduced in section 3. Some of GPU characteristics and our implementation of LS and DTW are described in section 4. Experimental results and analysis are covered in section 5. We conclude this paper and address directions for future work in section 6

II. FUNDAMENTALS OF LS AND DTW

A. Linear Scaling

Linear scaling is a simple yet effective method to compare the singing/humming clips to the database songs. Since the key and tempo of the query input may be different from those of the database songs, it is necessary to deal with key transposition and tempo variations for better matching. For key transposition, we can simply shift the mean values of the query pitch vector and the intended songs in database to the same level, such as zero. For dealing with tempo variation, we can simply apply linear scaling to the input pitch vector for comparison. Suppose that the input pitch vector is d-second long, we can then compress or stretch the vector to obtain r variants with lengths equally distributed between $m_1 \times d$ and $m_2 \times d$ where, m_1 and m_2 are the minimum and maximum scaling factors, respectively. The distance between the input query and a particular song is then defined as the minimum distance between these r vectors and a song. In practice, we usually use L1-norm to measure the distance. Fig. 1 shows an example, where we compress or stretch the d-second input query to obtain 6 vectors with lengths equally spaced between

0.8× d and 1.8× d. The best match is obtained when the scaling factor is 1.6.

Since linear scaling is very efficient both in its computation and in the one-shot way to handle key transposition, we shall adopt it as the first stage for the proposed two-stage comparison method.

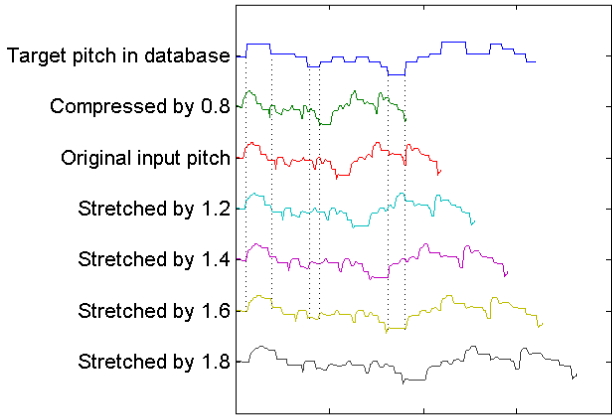

Fig. 1 An example of scaled query pitch for linear scaling

B. *Dynamic Time Warping*

Dynamic time warping [11] is another commonly used method for frame-based approach to QBSH. While LS scales the pitch vector uniformly, DTW is able to scale the pitch vector non-uniformly as it see fit during comparison, based on the concept of dynamic programing. The recurrence relation used in our system is shown next:

$$D(i,j) = |t(i) - r(j)| + min\begin{cases} D(i-1,j-2) \\ D(i-1,j-1) \\ D(i-2,j-1) \end{cases} \quad (1)$$

where the input pitch vector is $t(i), i = 1, \dots, m$, the reference pitch vector is $r(j), j = 1, \dots, n$ and $D(i,j)$ is the minimum distance starting from the left-most side ($i = 1$) of the DTW table to the current position (i,j), as shown in Fig. 3. Since the legal local paths have elevation angles of 27°, 45°, or 63°, we call this type of DTW as "DTW with 27°-45°-63° path."

The major advantage of DTW is its high accuracy due to its robustness in dealing with non-uniform tempos. Moreover, for DTW with 27°-45°-63°path, it can also tolerate sporadic unlikely pitch values (due to errors in pitch tracking) by skipping them directly. On the other hand, the major disadvantage of DTW is its requirement for massive computation, which is worsen by the fact that there is no one-shot scheme for dealing with key transposition. Thus we need to resort to binary-like search for finding the best pitch shift for key transposition.

DTW with key transposition is very time-consuming based on CPU architecture. Wang [2] has proposed a QBSH system using LS based on GPU architecture, and Kuo [6] has also parallelized DTW on GPU architecture. In this paper, we shall propose a two-stage method that can generate a better accuracy for QBSH, as described next.

Key Transposition Algorithm
k: the mean of input pitch vector
s: the one-side search range of key transposition
shift *k* to [*k-s, k, k+s*]
find the distance D_{k-s}, D_k, D_{k+s} using DTW
$M \leftarrow \min(D_{k-s}, D_k, D_{k+s})$
$s \leftarrow s/2$
$k \leftarrow k$ *where k corresponds to M in* [*k-s, k, k+s*]
for each *key transposition*
  shift *k* to [*k-s, k+s*]
  find the distance D_{k-s}, D_{k+s} using DTW
  $M \leftarrow \min(D_{k-s}, D_{k+s})$
  $s \leftarrow s/2$
  $k \leftarrow k$ *where k corresponds to M in* [*k-s, k+s*]
end

Fig. 2 Key transposition algorithm

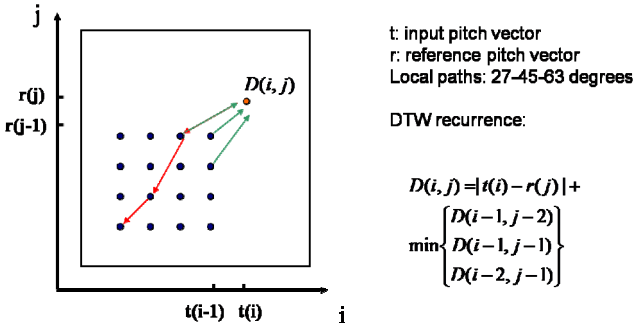

Fig. 3 The path of dynamic time warping

III. IMPROVED METHODS

A. *Two-stage Recognition and Candidate Song Selection*

The advantage of LS is its fast speed. On the other hand, DTW is slower, but it can achieve higher accuracy. To preserve the advantages of both methods, we choose to use LS for the first stage and DTW for the second stage. In this way, LS is first performed to select top-C candidate songs. Note that the number of candidate songs is a tradeoff between LS and DTW. The bigger C is, the higher the accuracy will be, but the computation time is also becoming longer.

At the second stage, DTW is used to compute the new ranking of the candidate songs. Here we adopt a weighting term in the recurrence relation, as shown in (2):

$$D(i,j) = |t(i) - r(j)| + min\begin{cases} D(i-1,j-2) \times C_2 \\ D(i-1,j-1) \times C_1 \\ D(i-2,j-1) \end{cases} \quad (2)$$

Fig. 4 shows an example of the effect of different weights. In particular, if C_2 is higher, the 27°-path will be more likely to be selected than other local paths. This can effectively

avoid the undesirable situation where the path DTW walks through contains some noise.

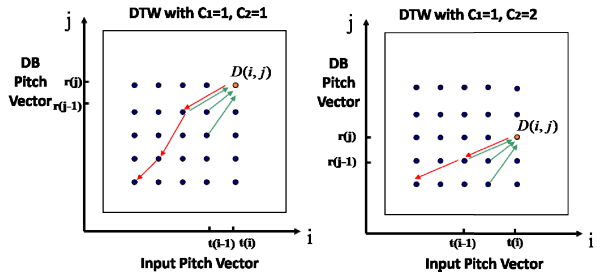

Fig. 4 The effect of different values for C_2.

B. *Combination of Melody Recognition*

Once we obtain the ranking by DTW at stage 2, we can combine the DTW-ranking with LS-ranking in order to achieve a better accuracy. Borda Count [9] is a single-winner election method which determines the winner by giving each candidate a score corresponding to the position in which it is ranked by each voter. Once all votes have been counted, the candidate with the most scores is the winner. We use Borda Count to determine the final ranking by computing a score that uses both rankings of DTW and LS, as shown next:

$$D_k = \sum_{i=1}^{M}(R - r_{ik}), r_{ik} = \begin{cases} r_{ik} - 1, \text{if } r_{ik} \le T \\ T, \text{if } r_{ik} > T \end{cases} \quad (3)$$

where R is the number of candidate songs, M is the number of different ranking methods, r_{ik} is the rank which is the k-th result order in the i-th melody recognition method. Finally, we will sum up the score obtained by different methods. The highest D_k means the most similar song it is in the database. The constant T constrains the score each candidate song will get from the rank, so it can avoid the situation that the correct answer is in top ranking at stage 1 but has a bad place at stage 2. Table I shows an example where constant T is 10 and R is 500 that each query can find the song in first or second place after using Borda Count method.

Table I Borda Count example with T = 10, R = 500, LS scaling factor: 80%-170%, resolution: 31 times, step: 3%, DTW key transposition 6 times

Query Song	LS Rank / Score	LSDTW Rank / Score	D_k (total score)	Final Rank
K 歌之王	1/500	389/490	990	1
一顆流星	1/500	37/490	990	1
世界唯一的你	350/490	1/500	990	2
愛如潮水	345/490	1/500	990	2

C. *Anchor Note Matching*

For both LS and DTW, it becomes very time-consuming if the comparison can starts from any note of a database song. If fact, a user usually sing or hum from the beginning of a phrase, not from any note in a song. Fortunately, our song database is original for karaoke and we do have the information of phrase locations. As a result, we use the first note in each phrase as the anchor note for comparison. This can reduce the computation significantly. Fig. 5 shows a typical example of 4 positions of anchor notes, together with the corresponding Mandarin lyrics.

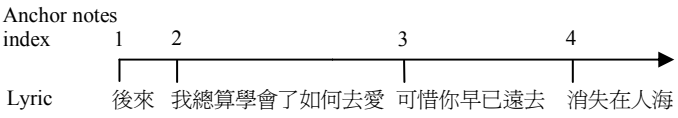

Fig. 5 The example of anchor note matching

IV. Implementation over GPU

A GPU consists of several streaming multiprocessors (SMs), each of which is composed of streaming processors (SPs), on-chip shared memories and registers. A GPU also contains global memory, constant memory and texture memory shared by SMs. The access time of global memory is larger, but the space is much larger than those of constant and texture memories.

Compute Unified Device Architecture (CUDA) [12] proposed by NVIDIA Corp. is a new way to accelerate scientific computing. Programmers can define C functions and execute them in parallel by thousands of threads which are the basic units in GPU. Several threads are grouped in a block, and several blocks are grouped in a grid. Data in shared memory can be accessed rapidly and shared by all threads within a block. This structure is shown in Fig. 6 [12].

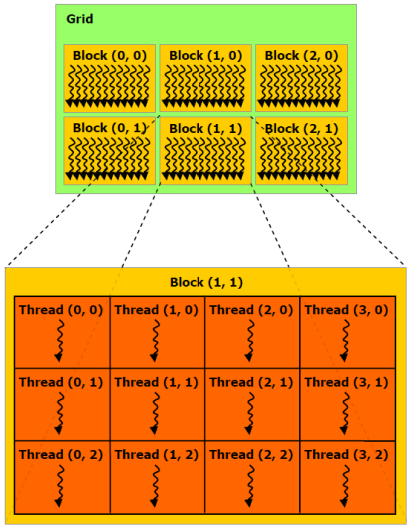

Fig. 6 The structure of block and thread

In our implementation, we first expand the music notes in the database to frame-based pitch vectors and move them to the global memory. To compress or stretch the input pitch vector for LS in stage 1, we launch r threads for each different scaling factor. After obtaining the scaled pitch vectors, we move them back to the main memory first, and then to the

constant memory for fast access. For comparison, we have one block for comparing one song, and each block has b threads. The computation tasks (which start from different anchor notes of a song) are equally distributed to the b threads. For DTW in stage 2, we launch one block for comparing one song. The DTW table is filled in parallel, where elements in a row are computed and filled simultaneously. We also need the shared memory for communicating with threads in the same block. After obtaining the distance between the query and each song in database with LS or DTW, we sort all the distances on CPU to obtain the top-n list.

V. Experimental Results and Analysis

The test corpus used in our experiment contains 2183 clips, corresponding to 964 popular songs. To increase the complexity of computation, we added 7467 noise songs to the database, such that the total number of songs in the database is 8431. The anchor notes of each song in the database are already tagged since the database is originally for karaoke, which has phrases clearly defined for synchronous lyrics display. Our platform is a PC with Xeon E3-1230 processor, 8GB DDR-3 1600 memories, and NVIDIA GeForce 660Ti GPU.

In our experiment, the LS scaling factor was varied from 0.8 to 1.7 to obtain 31 compressed or stretched versions of the original query input vector. Moreover, the frame size is 256 points with no overlap; the sample rate is 8 KHz, leading to a pitch rate of 31.25/sec.

The number of threads in a block affects the computation time of LS. Thus we want to find the best number of threads. Fig. 7 suggests that the minimum computation time can be obtained with 64 threads in a block. If the number of threads in a block is 32 or 64, the difference of computation time is not significant. But if we have more threads, the computation time is becoming higher since the resource in the SM is over utilized.

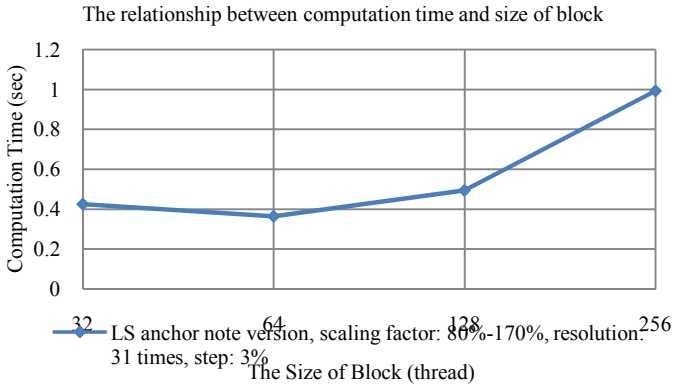

Fig. 7 The relationship between computation time and size of block

The most important parameters of DTW are key transposition range s and the path weight C_1, C_2. We shall first discuss the effect of key transposition range. As shown in Fig. 8, when the range is between -3 and +3, the accuracy is the highest regardless of the no. of key transposition. Thus we shall use [-3, +3] as the key transposition range in the following experiments.

Another parameter set for DTW is the weights $[C_1, C_{2,}]$ defined in (2). To tune the parameters efficiently, we employ both brutal force and downhill Simplex search [13] on the proposed two-stage (LS+DTW) method with Borda Count. The number of candidate songs R was set to 50 while the candidate song threshold T was set to 10 for this experiment. We explored three sub-experiments:

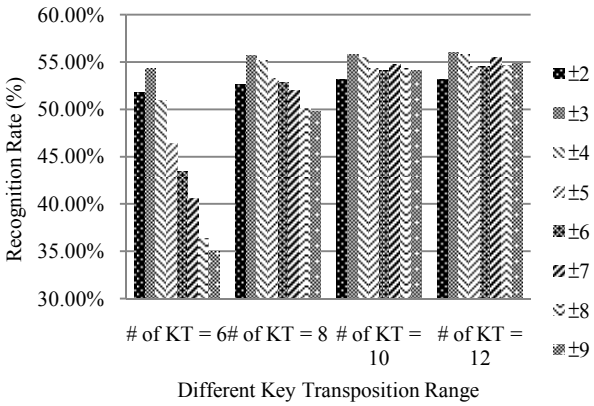

Fig. 8 The different key transposition ranges in dynamic time warping with 2183 queries. (KT = key transposition)

1. We fixed C_1=1 and varied C_2 to obtain the result in Fig. 9. The best accuracy is obtained when C_2=1.4.

2. We fixed C_2=1 and varied C_1 to obtain the result in Fig 10. The best accuracy is obtained when C_2=1, but the accuracy is not as good as the best on in sub-experiment 1.

3. Starting from the best case in sub-experiment 1, we tried downhill Simplex search to obtain the result in Fig. 11. It turns out that the best case in sub-experiment is the best performance obtained in this sub-experiment.

Thus in the following experiments, we shall set $[C_1, C_{2,}] = [1, 1.4]$.

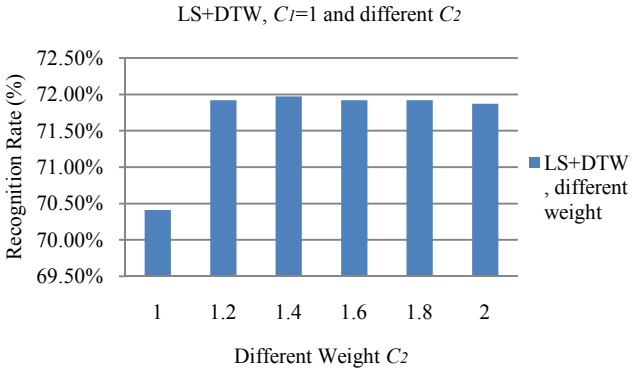

Fig. 9 Accuracy vs. different values for C_2 in LS+DTW, with C_1= 1, no. of key transpositions is 7, and the number of candidate song is 50.

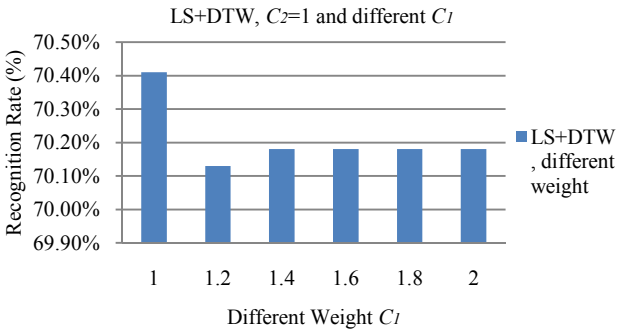

Fig. 10 Accuracy vs. different values for C_1 in LS+DTW, with C_2 = 1, no. of key transpositions is 7, and the number of candidate song is 50.

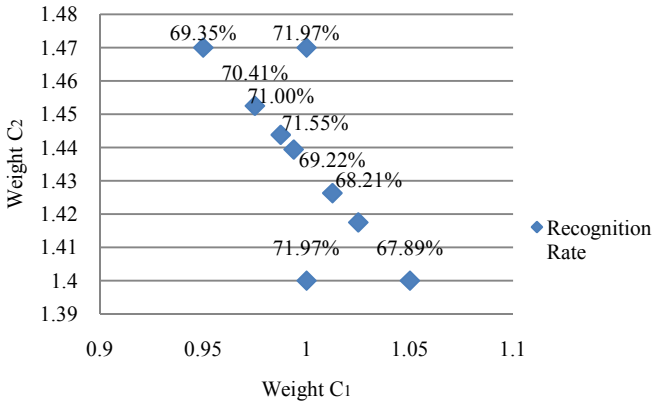

Fig. 11 Accuracy vs. different values of C_1, and C_2 in LS+DTW, with no. of key transposition is 7, and number of candidate song is 50

To find the best parameter T in (3), we perform a brutal force search to obtain the result in Fig. 12. The best performance is obtained when T is 10, which will be used throughout all the other experiments in this paper.

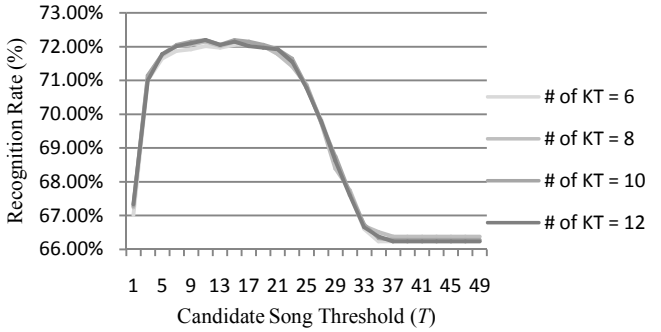

Fig. 12 Accuracy vs. different values of T, with number of candidate song is 50. (KT = key transposition)

Fig. 13 shows the performance comparison between family A (LS+DTW with anchor notes) and family B (without using anchor notes). Both families do not use Borda count for reranking. Obviously the use of anchor notes can significantly reduce the computation time by almost 2 seconds per query. However, the accuracy is a little bit lower, which could be due to the imprecise anchor notes in the dataset. For the rest of the experiments, we shall use anchor notes for matching.

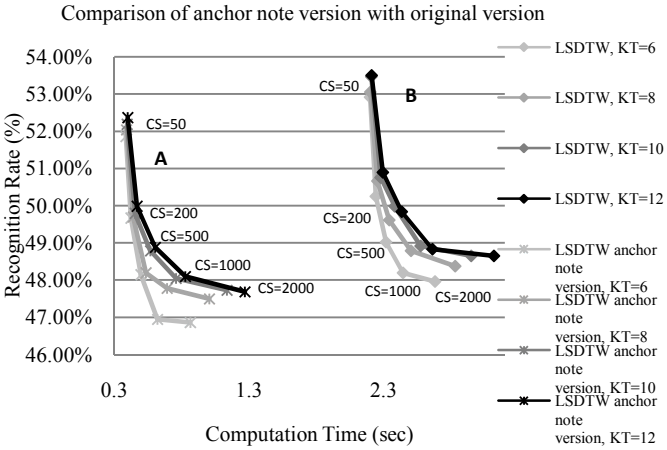

Fig. 13 The performance comparison between the anchor note version (denoted as family A) and the original version (without using anchor notes, denoted as family B). Both are not using Borda Count. (KT = key transposition, CS = candidate song)

In the next experiment, we shall compare the performance before and after parameter tuning for DTW and key transposition. As shown in Fig. 14, we have two families of curves. Family A uses a set of optimal parameters, with C_1=1, C_2 =1.4, and key transposition range is [-3, +3]. On the other hand, family B uses a set of non-optimal parameters, with C_1 =C_2=1, and key transposition range is [-1.5, +1.5] (which follows [6]). It is quite obvious that the performance of family A is about 7% higher than that of family B. Moreover, we can see that if DTW is not optimized at all, its performance will not be as good as LS. This is revealed in family B, where more candidate songs for DTW leads to a even worse performance.

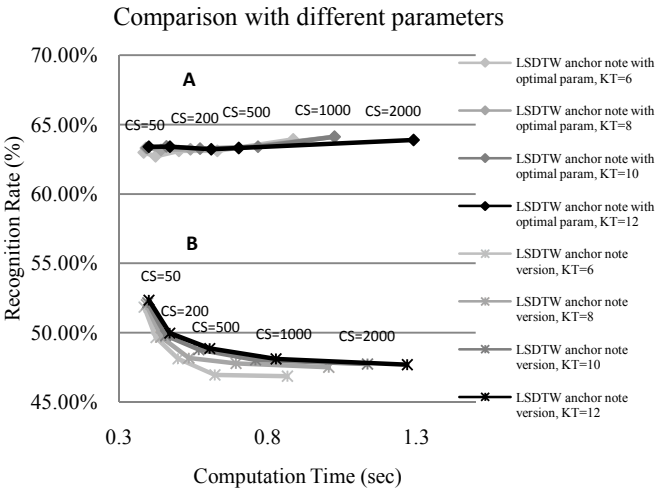

Fig. 14 The performance comparison between optimal set (family A) and non-optimal set (family B) of parameters. (Both do not use Borda count for reranking.) Please refer to the text for the parameter values used in both families.

In the next experiment, we shall use the Borda count for reranking and repeat the previous experiment. Fig. 15 demonstrates the results after using the Borda count for reranking. Apparently the use of Borda count for reranking has a dramatic effect on boosting the accuracy.

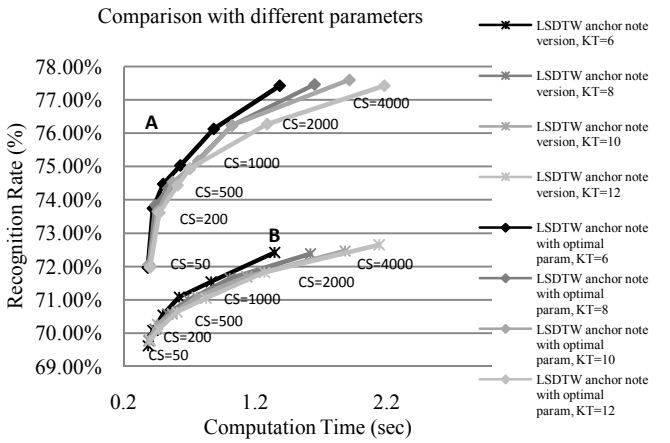

Fig. 15 The performance comparison after using the Borda count for reranking. All the other settings are the same as in Fig. 14.

We can also observe that the number of key transpositions does not affect the accuracy too much, but the number of candidate song does affect the accuracy a lot. In particular, when the number of candidate songs is increased from 50 to 4000, the accuracy is also increased from 72.01% to 77.65%.

The overal comparison in Fig. 16 demonstrates that the proposed two-stage method improves the accuracy up to 10%. Moreover, the computation time is increased only marginally, slightly greater than LS but much less than DTW.

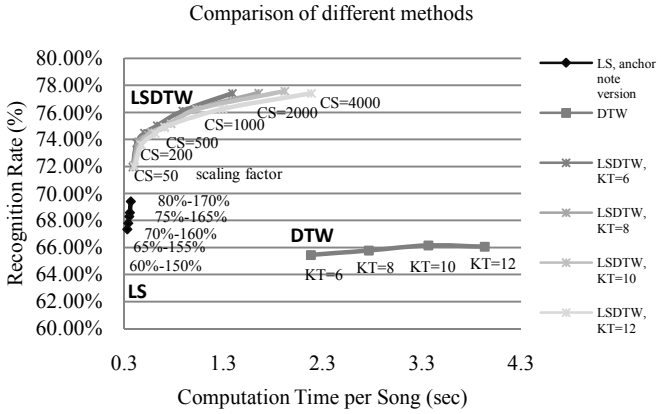

Fig. 16 Performance comparison among LS, DTW, and the proposed two-stage method (denoted as LSDTW).

VI. Conclusions and Future Work

In this paper, we have proposed a two-stage method that combines the results of LS and DTW to perform reranking. The speedup factor is about 7 (compared with DTW alone on GPU), and the response time has been reduced from 2.3 seconds to 0.3 seconds. The recognition accuracy has reached 77.65% which is better than LS and DTW alone.

One of our immediate tasks is to investigate the potential of using more than two methods for reranking. Moreover, we shall also try to find other better ways for score combination based on machine learning methods, such as learning to rank.

References

[1] J.-S. R. Jang, J.-C. Chen and M.-Y.Kao."MIRACLE: A Music Information Retrieval System with Clustered Computing Engines," ISMIR 2001.

[2] C.-C. Wang, C.-H.Chen, C.-Y.Kuo, L.-T.Chiu, and J.-S. R. Jang, "Accelerating Query by Singing/Humming on GPU: Optimization for Web Deployment," ICASSP 2012

[3] G. Poli, A. L. M. Levada, J. F. Mari, J. H. Satio, "Voice Command Recognition with Dynamic Time Warping (DTW) using Graphics Processing Units (GPU) with Compute Unified Device Architecture (CUDA)," in Proceedings of the 19th International Symposium on Computer Architecture and High Performance Computing , SBAC-PAD 2007, Brazil, pp. 19–25, 2007.

[4] Jun Li, Shuang ping Chen, Yan hui Li, "The Fast Evaluation of Hidden Markov Models on GPU," in IEEE International Conference on Intelligent Computing and Intelligent Systems, Shanghai, vol. 4:426-430, Nov., 2009.

[5] P. Ferraro, P. Hanna, L. Imbert, and T. Izart, "Accelerating Query-by-Humming on GPU" in Proceedings of the 10th International Conference on Music Information Retrieval, ISMIR 2009, pp. 279–284, 200.

[6] Chin-Yang Kuo, "Accelerating Query By Singing/Humming on GPU," National Tsing Hua Univ. 2013.

[7] Yi-Fan Fang, "Improvement and Implementation of Query by Singing/Humming Systems,"National Tsing Hua Univ. 2010

[8] Tzu-Chiao Lin, "Research and Implementation of Query by Singing/Humming for Embedded Karaoke Systems," National Tsing Hua Univ. 2009

[9] Tin Kam Ho, J. Hull, Sargur N. Srihari, "Decision Combination in Multiple Classifier Systems," in IEEE Transactions on Patter Analysis and Machine Intelligence (PAMI), Jam., 1994

[10] Ming-Xian Zou, "Query By Singing/Humming Using Combination of Classifiers," National Tsing Hua Univ. 2008.

[11] Jang, J.-S Roger, Ming-Yang Gao, "A Query-by-Singing System based on Dynamic Programming," International Workshop on Intelligent Systems Resolutions (the 8th Bellman Continuum), PP. 85-89, Hsinchu, Taiwan, Dec 2000.

[12] NVIDIA, "NVIDIA CUDA C Programming Guide Version 4.2."

[13] Lagarias, J.C., J. A. Reeds, M. H. Wright, and P. E. Wright, "Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions," SIAM Journal of Optimization, Vol. 9 Number 1, pp. 112-147, 1998