SoundCompass: A Practical Query-by-Humming System

Normalization of Scalable and Shiftable Time-Series Data and Effective Subsequence Generation

Naoko Kosugi, Yasushi Sakurai, Masashi Morimoto
NTT Cyber Space Labs. 1-1, Hikarinooka, Yokosuka, Kanagawa, Japan
{kosugi.naoko, sakurai.yasushi, morimoto.masashi}@lab.ntt.co.jp

ABSTRACT

This paper describes our practical query-by-humming system, *SoundCompass*, which is being used as a karaoke song selection system in Japan. First, we describe the fundamental techniques employed by SoundCompass such as normalization in a time-wise sense of music data, time-scalable and tone-shiftable time-series data, and making subsequences for efficient matching. Second, we describe techniques to make effective feature vectors based on real music data and do matching with them to develop accurate query-by-humming. Third, we share valuable knowledge that has been obtained through month's of practical use of SoundCompass. Fourth, we describe the latest version of the SoundCompass system that incorporates these new techniques and knowledge, as well as describe quantitative evaluations that prove the practicality of SoundCompass. The new system provides flexible and accurate similarity retrieval based on k-nearest neighbor searches with multi-dimensional spatial indices structured with multi-dimensional feature vectors.

1. INTRODUCTION

A vast amount of information flows through the WWW, and all of it can be quickly accessed via search engines such as Google. Even so, there is no good way to search for a song that people know only a part of!!

These days, recorded music is everywhere we go, in shops, in cafes, etc., and it is almost impossible to find TV programs or commercials that do not include music. Music seems to be an essential part of our lives. It is no wonder that virtually everyone has probably heard a familiar song whose name they couldn't remember. Wouldn't it be great if you could "remember" the name by only humming a part of the melody? A "query-by-humming system" makes this possible.

The problem of searching for a song that you remember only a part of would seem to be a comparatively simple one. It is not, however, because there are many issues that require a wide variety of technologies to resolve; they are as follows:

- A Similarity retrieval technique is necessary.
  People hum and input a part of a melody as a query with a microphone. Some errors may be included when the voice signal data is processed. Moreover, few people can sing a song in exactly the same way as written in the music score.
- Complex time-series data handling is necessary.
  Music can be recognized even if it is performed faster or slower than usual, or in higher or lower keys. This is because music is time-scalable and tone-shiftable time-series data.
- A Complex subsequence matching technique is necessary.
  When users search for a song by humming, no one knows in advance what segment they will hum, or how long, how fast, or in what key they will hum the tune.
- A Compact database, efficient indices, and fast matching techniques are necessary to make a practical system.

A query-by-humming system is not only essential for music searching; it is also a noteworthy development for the database research area as a whole because it incorporates a number of very important database technologies, such as similarity retrieval, time-series data handling, and subsequence matching. Our previous version of SoundCompass [8] has been used as a part of a karaoke song selection system in Japan since the end of 2000. In this paper, we describe techniques the latest version uses to time-normalize music data, which is time-scalable and tone-shiftable time-series data, and to do efficient subsequence matching for efficient query-by-humming. We also confirm the practicality of the new techniques through quantitative evaluations.

Figure 1 shows the meaning of the terms "time-scaling", "tone-shifting", and "time-normalization" used in this paper. Here, time-scaling means "scaling in a time-wise sense", tone-shifting means "shifting in a tone-wise sense", and time-normalization means "normalization in a time-wise sense".

2. RELATED WORK

Notes are used in both note-based [4, 9] and beat-based processing [8]. Recently, though frame-based music data processings that directly use voice signals have been proposed [1, 5, 10, 12]

The advantage of the frame-based processing is that no errors caused by note transcription are included in the queries. However, it is not always true that through frame-based processing better queries can be made, because it is inevitable that many errors sources, such as noise, affect humming data. In addition, timing information remains in the data, and this will influence matching. Thus, some research uses *Dynamic Time Warping* [6, 11], or *DTW*, for matching to reduce the influence of the time elasticity of music data [5, 12]. DTW was originally used in speech recognition, since it enables matching without the influence of the local timing

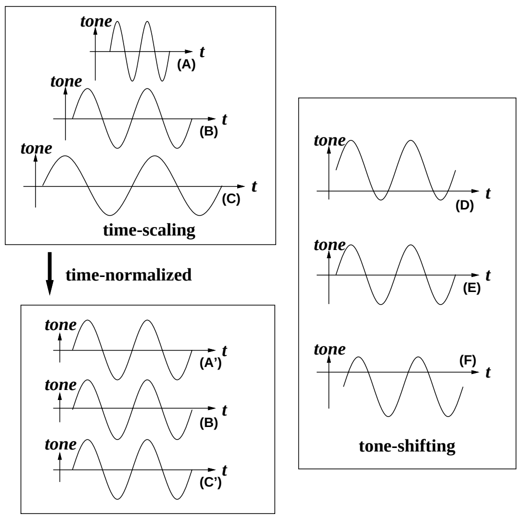


Figure 1: Time-scaling, tone-shifting, and time-normalization of time-series data.

gaps in data. Indeed, DTW seems to be useful for matching of music data and humming, because general people are not good at keeping a constant tempo when they hum. However, its advantages and disadvantages for query-by-humming systems are not clear enough, since its application is limited at present.

3. TIME-NORMALIZATION AND GENERATION OF SUB/SUBSUBSEQUENCES

In this section, we describe methods to time-normalize music data and make sub/subsubsequences from them. We used 21,804 songs and 591 hummings for our investigations and evaluations. All the songs are in MIDI format and used in karaoke. Most songs are Japanese pop hits; however, about 6,000 of them are foreign songs, simple nursery songs, and folk songs. Thirty-four people (26 males, 8 females) hummed the samples. Some hummed in time with a metronome, and others without one.

3.1 Onset-to-onset Duration

For the hummings, the duration of an utterance is defined as the interval from the time one utterance starts to the time the next utterance starts, and for songs, the duration of a note is defined as the interval from the time one note starts sounding to the time the next note starts sounding. This is called the onset-to-onset duration [9]. We refer to the onset-to-onset duration of an utterance as an *utterance-duration* for hummings, and that of a note as a *tone-duration* for music data.

Why we focus only on the start timings of utterances and notes is that it is highly likely that the timing at which an utterance starts may often be temporally correct. If the utterance starts with the wrong timing, the humming lacks rhythm, so it is difficult for people to recall the correct song. Thus, many people are especially sensitive to the start timing of the utterances as well as the pitches when they hum. The timing when an utterance starts can be fairly accurately found based on changes in amplitude of the voice signal.

3.2 Unit-length and Beat-resolution

Song data and hummings are time-normalized by converting all tone-durations and utterance-durations to relative values based on the most frequently appearing durations. We define the basis for hummings as *unit-length* (ul), and for split song data (sub/subsubsequence in this paper) as *beat-resolution* (br). Let L_{h_i} be the i-th utterance-duration, and L_{s_j} be the j-th tone-duration; they are converted into L'_{h_i} and L'_{s_j} with the following equations.

$$L'_{h_i} = L_{h_i}/ul \qquad L'_{s_j} = L_{s_j}/br$$

3.3 Subsequences and Subsubsequences

We propose a two-stage music data split, in other words, generation of subsequences and subsubsequences. Segments that hold the same amount of music information are necessary for efficient subsequence matching. However, it is difficult to define adequate "segments" for music data because of the data's time-elasticity. Thus, we propose a two-stage data split; the first is a temporal split to define the segments, and the second is a split in which every segment has the same amount of music information based on the beat-resolution. The temporal split is used to create *subsequences* and the second split is used to make *subsubsequences* from a subsequence.

Let the most frequently appearing tone-duration in each song be the temporal beat-resolution (TBR) of the song. The song data is temporally normalized with TBR, and then subsequences of constant length based on the TBR are generated by using the sliding-window method [2]. Next, for all subsequences, the most frequently appearing tone-duration in every subsequence is reselected. If the tone-duration of a subsequence is not the same as the TBR, it is defined as the real beat-resolution (RBR) of the subsequence. Then, both slide-length and window-length are reset based on the RBR, and subsubsequences are generated from the subsequence. Figure 2 shows subsequences and subsubsequences for the cases RBR < TBR (left), RBR = TBR (middle), and RBR > TBR (right), respectively.

- RBR < TBR
  Subsubsequences whose lengths are equal to the window-length based on RBR are made by deleting music data from the end of the subsequence.
- RBR > TBR
  Subsubsequences whose lengths are equal to the window-length based on RBR are made by adding music data that come directly after the subsequence.

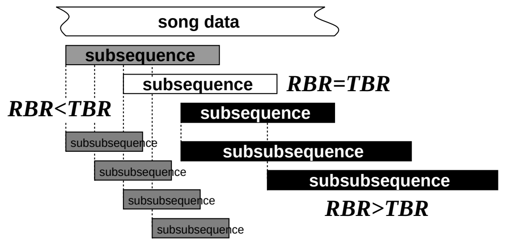


Figure 2: Generation of sub/subsubsequences.

Figure 3 shows the percentage of how often the most frequently appearing tone-duration appears in each song. The figure shows that the most frequently appearing tone-duration appears in the 40%–60% range in many songs. Thus, subsubsequences are also necessary in addition to subsequences to make all the segments be normalized based on the most frequently appearing tone-durations in them.

3.4 Phonetic Value Rate

Tone-durations in our MIDI data have a wide variation, in contrast to what we imagined would be a narrower variation. For example, a standard quarter note is represented with 480 ticks; however, quarter notes in our data correspond to 460, 500 ticks, etc. This is because MIDI data makers do their best to make values that are close to the

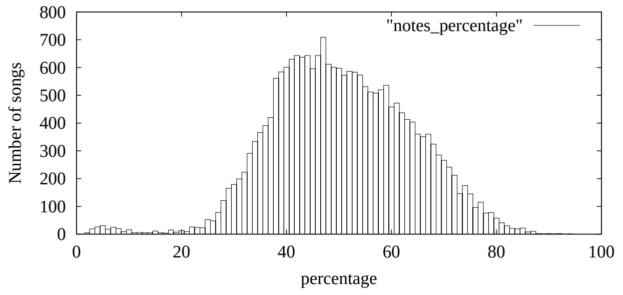


Figure 3: Percentage of the most frequently appearing tone-duration appearing in each song

original performance. However, few users can utter notes consistently reflecting on such small differences. In the same way as in humming, most of the variations in L'_{h_i}s are not intended by the user but are simply mistakes. This means it is not good for queries and data in the database to reflect such variations. Thus, we propose a method to map all tone/utterance-durations converted based on the unit-length and beat-resolution to representative values, *phonetic value rates*, in *pvr*s. The possible phonetic value rates were,

$$0.5, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 6.0\ pvrs \qquad (1)$$

These values are determined based on the frequency of appearance of converted tone-durations. Figure 4 shows the

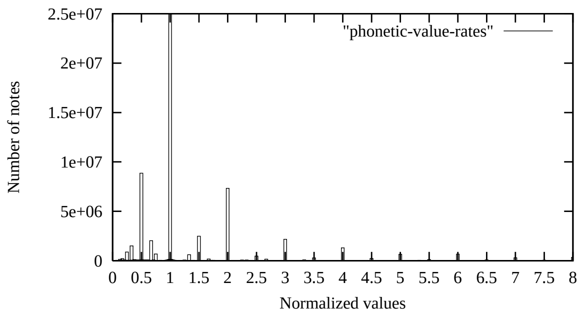


Figure 4: Distribution of L'_{s_j}.

distribution of L'_{s_j} of all sub/subsubsequences. The L'_{s_j}'s exceeding 8.0 are not displayed. There are many notes with 1.0, 0.5, or 2.0, and fairly many with 1.5, 3.0, 4.0, 5.0, or 6.0. It is difficult for casual singers to extend long notes correctly. Thus, the maximum phonetic value rate was set as 6.0 pvrs, and longer tone-durations are mapped to 6.0 pvrs.

Song data time-normalized with this method are similar to those of the beat-based processing whose effectiveness was already proved in our previous paper [8]. This method not only provides fast matching, but also improves the discriminatory power by utilizing the duration information of each note/utterance as well as pitches for matching.

4. FEATURE VECTORS AND MATCHING STRATEGY

4.1 Feature Vectors

4.1.1 Tone-in Phonetic Value Rate

One of the greatest pleasures of music appreciation is listening to timely changes in the heights and lengths of tones. These changes also constitute one of the most important features of music. As such, we propose a feature vector that represents the transition of tones in a timely fashion, which we call *Tone-in phonetic value rate*, or *TPV*. It is defined as tone values in a line that are ordered for every phonetic value rate. Each vector element is a difference relative to a given tone, which we call the *base-tone*; this makes it possible to search for songs with various key heights.

The followings explains how to make a TPV with the first bar of an example tune in Figure 5. C4, D4, and so on represent pitch names and the number to the right is the MIDI note number. The vector generated from the melody depends on the beat-resolution of the sequence that includes it. For example, if the base-tone is the lowest tone, 60 in this example, in the tune, when the beat-resolution is a quarter note, the first note has 3.0 pvrs and the next one 1.0 pvr. Thus a four-dimensional vector (0, 0, 0, 7) is generated. When the beat-resolution is an eighth note, the first note has 6.0 pvrs and the next one 2.0 pvrs. Thus, an eight-dimensional vector (0, 0, 0, 0, 0, 0, 7, 7) is generated.

4.1.2 Matching with Vector Heads

We proposed a "partial tone transition feature vector" for the previous SoundCompass [8] to match the heads of the vectors made from song data and from humming. We do so by setting the heads of feature vectors so that they don't have to be the beginnings of the sub/subsubsequences. This technique is used for both time-normalized hummings and song data.

Here, let the slide-length be represented *sl*, and the window-length *wl*. The feature vector is basically a TPV-type vector whose head is set where the most distinguishable tone that first appears within *sl* pvrs in a sub/subsubsequence. Thus, the dimension of the vector is $(wl - sl)$. We confirmed that the retrieval accuracy was the highest when the highest tone was used as the most distinguishable. This feature vector is called the *variable slide-length TPV*, or *VSTPV*. On the other hand, feature vectors whose heads are identical to the beginnings of the sub/subsubsequences are called *constant slide-length TPV*, or *CSTPV*.

The generation of VSTPVs and CSTPVs is explained below for the example tune shown in Figure 5. Suppose that

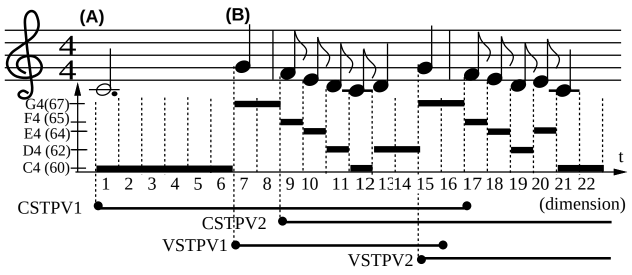


Figure 5: Tune in which the start beat of the beginning part (A) differs from that of the bridge part (B).

the bridge part (B) begins at the fourth beat of the first bar, and the start of the song (A) is at the first beat of the first bar. This means that this tune is an auftakt with respect to the bridge part. All of the beginnings of the sub/subsubsequences made from this tune are at the first beat of each bar if the beat-resolution is an eighth note and the slide-length is 8 pvrs. Thus, the two CSTPV vectors' heads are the same as the beginning of each bar. This means that when a user hums the bridge part, there is no CSTPV vector in the database whose head is consistent with that of the hummed tune. On the other hand, if we let the most distinguishable tone be the highest tone, the head of the VSTPV1 is the fourth beat of the first bar (7th element of CSTPV1) and matches the head of the bridge part. Thus, it doesn't matter whether the user hums from the start of the song (A) or from the bridge part (B); the VSTPV vector made from such hummings will be the same one, and there will be a matching VSTPV vector in the database. In pilot studies, VSTPV showed almost the same retrieval accuracy as did CSTPV whose slide-length is 1/4 that of the VSTPV.

4.2 Matching Strategy

4.2.1 OR-retrieval among Humming Subsequences

If a user hums longer than the window-length, the sliding-window method will generate multiple subsequences. We use each subsequence for matching and the final result is generated by merging the results from each matching.

Suppose that m subsequences are generated from a humming h, and n subsequences are generated from a song s. The distance between the song and the humming, $D(h, s)$, is defined by eq.(2). The distance between the i-th subsequence of a humming h and the j-th sub/subsubsequence of a song s is represented as $d(h_i, s_j)$.

$$D(h, s) = \min_{1 \le i \le m, 1 \le j \le n} \{d(h_i, s_j)\} \quad (2)$$

This is effective, because we do not know in advance what errors exist where in the humming nor which subsequence is the most discriminable among them.

4.2.2 AND/OR-retrieval of Features

The percentage of hummings whose CSTPV heads were not matched to those made from song data but whose VSTPV heads were matched to those made from song data was about 24.6%. On the other hand, the percentage of hummings whose VSTPV heads were not matched to those made from song data but whose CSTPV heads were matched to those made from song data was about 5.8%. Thus, it will be effective to use both features separately for the retrieval and use the better one, i.e., OR-retrieval.

Another type of feature vector is necessary to deal with the number of utterances/notes, since the TPV-type feature vectors can not distinguish between eight successive eighth notes and one whole note in the same pitch. Thus, a *distribution of tone difference*, or *DTD* [8], which represents the tone difference between successive notes is introduced. In the pilot study, we confirmed that it is useful as an auxiliary feature vector to consider the number of utterances and notes, though it does not have sufficient discriminatory power by itself. Since DTD is not exclusive, it is used in AND-retrieval with VSTPV and CSTPV. In Section 6, the retrieval accuracy is evaluated with the feature combination "(VSTPV && DTD) || (CSTPV && DTD)".

5. SOUNDCOMPASS

5.1 Database Construction

Database construction starts with extracting melody data from 21,804 songs. Time-normalization and generation of sub/subsubsequences are performed simultaneously as described in Section 3. The slide-length and the window-length are set to be 8 pvrs and 32 pvrs, respectively, since the most frequently appearing phonetic value in our song data is an eighth note. This means 1.0 pvr may correspond to an eighth note in many songs. Moreover, 97% of all songs are in 4/4 time, and many phrases in 4/4 time are about four bars. A bar and four bars in 4/4 time means 4 beats and 16 beats, in other words, 8 eighth notes and 32 eighth notes, respectively.

Next, VSTPVs, CSTPVs, and DTDs, are generated. The VSTPVs and CSTPVs are tone-shifted to let their average tone value be 0.0. Here, sub/subsubsequences that produce the same feature vectors are deleted, letting the first appearing one be kept, in order to reduce the size of the database and to keep the system's discriminatory power high. (Most of the songs have redundant structures [3], i.e., several verses [8].) Finally, feature vectors are individually indexed according to the feature.

5.2 Hummed Tune Processing

A hummed tune is recorded in 11kHz/8bit/monoral. The acoustic data is processed with 512-point data for every 256 points by FFT to pick up the timing of utterances and to track the fundamental frequency (F0) of each frame. In SoundCompass, pitch and duration of an utterance are determined based on the assumption that one utterance corresponds to one note. Since many melodies that have lyrics allocate one syllable (one letter in Japanese) per note, the assumption seems valid.

Next, the unit-length is determined, and each utterance-duration is adjusted with a set of phonetic value rates. Subsequences are then generated. Feature vectors, VSTPVs, CSTPVs, and DTDs, are generated from each subsequence and used for queries.

5.3 Retrieval Engine and Similarity Measurement

To achieve scalability and high-speed similarity retrieval, we built a search engine with distributed indices, which we call *Keren* [7]. All queries are processed in parallel with multiple threads that are generated when Keren is initialized. Keren comprises a global manager (GM) and multiple database managers (DM). GM accepts retrieval requests from clients and sends queries to DMs. It also receives retrieval results from DMs, merges and formats them, and sends them to the clients. In the database update phase, the GM evenly distributes input songs to DMs and requests database update. The DMs, in the retrieval phase, accept queries from the GM, retrieve results, and send them to GM. In the update phase, DMs process data distributed by GM and restructure their indices so as to make them well-balanced. This ensures that Keren is able to conduct efficient searches.

The similarity retrieval function finds vectors in each feature's vector space that are close to the vectors generated by the hummed tune. The city-block distance is used for the similarity measurement. The shorter the distance to the sub/subsubsequence is, the more the sub/subsubsequence resembles the humming. All results from each feature vector space are merged based on the combination described in Section 4.2.

5.4 User Interface

The hummed tune is recorded through a microphone. The user is required to clearly hum the song notes using only the syllable "ta" to transcribe pitches as accurately as possible.

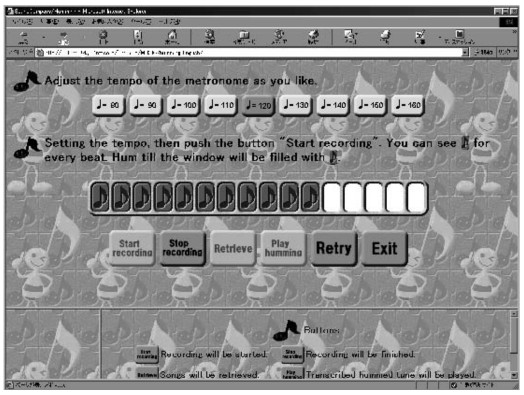
Figure 6 shows the GUI screen for recording humming. Here, SoundCompass is in the middle of recording (the screen's central bar is still not completely filled with note icons). Explanations for each button can be seen in the lower part of the screen. Thus, beginners can easily use the system by referencing them.

5.5 Issues

Here, we discuss issues that have been brought up before in many papers, but which we believe we should discuss again, in view of practical use of SoundCompass.

5.5.1 Is a metronome necessary?

The previous version of SoundCompass [7, 8] needed a metronome to time-normalize the humming data. The SoundCompass does not need one because song data and hummings are time-normalized automatically. Nonetheless, some people say it's better to sing to metronome beats because



Adjust the tempo of the metronome as you like.

Setting the tempo, then push the button "Start recording". You can see ♪ for every beat. Hum till the window will be filled with ♪.

Recording will be started. Recording will be finished.

Songs will be retrieved. Transcribed hummed tune will be played.

Figure 6: Recording Screen for Humming

a metronome lets people recognize the recording time, and its beats urge smooth humming. Moreover, some people say that they can hum easier to the beat of the metronome. Thus, the new SoundCompass system still has a metronome as an optional user interface. Users can retrieve songs even if they are out of beat with the metronome.

5.5.2 Is it better to sing to lyrics than with "ta"?

Some systems, including SoundCompass, require users to hum with only a syllable "ta" or "da", because it is effective for voice signal processing [8, 9]. In addition, we found other advantages.

A fairly large number of people said that singing to lyrics is more embarrassing than singing to the syllables "ta" or "da". Moreover, many people said that actual humming to a microphone is far more embarrassing. They said singing with "ta" tends to eliminate embarrassment as they get used to it. The reason may be that they might not want to be embarrassed in front of other users by singing poorly to lyrics that have been recorded by popular singers.

Even more important is that few users can concentrate on the pitch and duration of each note when they sing lyrics. The result will be poor-quality queries and failure to retrieve correct songs. Music class teachers often sing with the syllable "ta" in teaching their classes, because it allows them to sing melody more correctly than they could by using the actual lyrics. Thus, if people really want to find the song, it's important for them to concentrate on reproducing the melody as correctly as possible.

The reason that the experimental subjects said it was better for them to sing lyrics is that they selected songs whose melodies and lyrics were already familiar to them. This doesn't necessarily correspond to how such a system would be used in practice. From a practical viewpoint, we should not consider a system that can accept singing to lyrics, but one that enables users to get used to singing with "ta" or "da" naturally.

5.5.3 Is there any other use for query-by-humming?

There are a variety of usages for query-by-humming systems besides searching for unknown songs. Some people use SoundCompass in our laboratory as a game and enjoy making "hits" or "misses" by setting a variety of rules for humming. Some people use it to check key differences between keys of their voice and those of songs for singing training. Some people use it to point to a specific place within a song, since it provides subsequence matching. For example, if lyrics are accompanied by melody data such as meta data, we can refer to any portion of lyrics with a query-by-humming system. Thus, we can conclude that a query-by-humming is a very useful application.

6. EVALUATION

6.1 Experimental environment

Figure 7 shows an experimental system. Keren (Section 5.3), SoundCompass Server (query-by-humming server), and Client PCs are connected to a network. A microphone is connected to a client PC. Humming is sent to the SoundCompass server. The server processes it, makes queries and sends them to Keren. Keren retrieves the queries and sends results to the SoundCompass server. The server receives the results, formats them, and sends them to the client PCs.

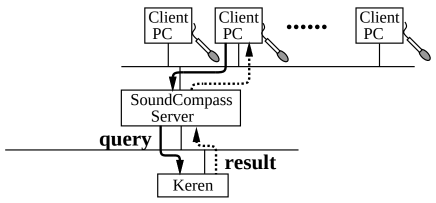


Figure 7: Experimental System

6.2 Effectiveness of Time-Normalization and Sub/subsubsequence Generation

The number of hummings that have sub/subsubsequences that are time-normalized in the same way as humming is shown in Table 1. "Time-normalized in the same way as a humming" for song data means the tone-durations in one of the sub/subsubsequences of the song data have the same phonetic value rates as those of the utterance-durations in the humming. As shown in Table 1, about 83.93% of hummings have subsequences that are time-normalized in the same way and about 9.98% of hummings have only subsubsequences that are time-normalized in the same way when only one unit-length is used for the time-normalization. This means subsubsequences are necessary. Moreover, by using the second-most frequently appearing utterance-duration (two unit-lengths) in addition to the first most, about 96.15% of hummings have sub/subsubsequences that are time-normalized in the same way as those in the database.

Table 1: Number of sub/subsubsequences time-normalized in the same way as humming

	number of unit-lengths	
	1	2
subsequence	496 (83.93%)	513 (86.81%)
subsubsequence	59 (9.98%)	61 (10.31%)
summation	555 (93.91%)	574 (96.15%)

6.3 Retrieval Accuracy

The effectiveness of the techniques proposed here is quantitatively evaluated by comparing the retrieval accuracies of three types of combinations of data-handling and matching method. The data were also split with a sliding-window to speed up matching for the DTW case. Figure 8 shows retrieval accuracies of the three types. The x-axis represents rank and the y-axis represents the retrieval accuracy by showing the percentage of songs retrieved within the rank. "DTW" indicates that DTW alone was used for matching. "TN+DTW" indicates that DTW was used for matching and music data was time-normalized based on the techniques described in Section 3. "SC" indicates that a city-block distance measurement was used for linear matching, music data were time-normalized, and the feature vectors and matching strategies described in Section 4 were applied.

Difference between "DTW" and "TN+DTW" is the way in which they handled the music data: the former uses timing information as is and the latter uses the time-normalized

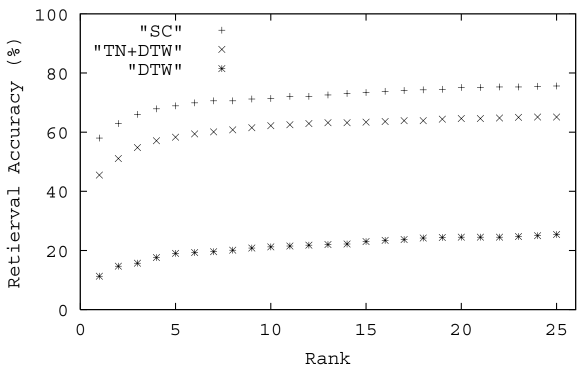


Figure 8: Retrieval Accuracy

timing information. Using the timing information as is means that it is highly likely that the amount of musical information in the song data and in the humming data will be different. Because two 10-seconds music data for the same part of the same piece of music can not be regarded as maintaining the same amount of information in musical sense if they are not played at the same tempo. Thus, the matching between them will not be fair. On the other hand, if the music data are time-normalized, 10-beats music data or 10-pvrs music data can be regarded as maintaining the same amount of information in the musical sense, even if their durations are different. Consequently, fair and fast matching becomes possible. This is the main reason for the difference in the retrieval accuracies between "DTW" and "TN+DTW". Thus, it is clear that equalizing the quantity of information for correct matching is necessary, i.e., the effectiveness of the time-normalization and generation of sub/subsubsequences is proved.

The reason that the retrieval accuracy of "SC" is better than that of "TN+DTW" is that; 1) linear matching raises the discriminatory power since both pitch and duration of notes can be taken into account for matching; 2) VSTPVs made from song data will match VSTPVs made from humming data more often than if CSTPVs were used; 3) the retrieval with multiple kinds of feature vectors also improves the discriminatory power.

In summary, we find two important points for music data matching as follows:

- The amounts of musical information in the two data sets to be compared must be equal. In other words, both the beginnings and ends of the music data must be identical.
- Note-duration information should also be used for matching in addition to pitch information to improve discriminatory power.

6.4 Discussion

Though the effectiveness of the techniques proposed in this paper has been quantitatively confirmed, the retrieval accuracy shown in Figure 8 is not commensurate compared to the accuracy of time-normalization and sub/subsubsequence generation shown in Table 1. The main reasons are errors in voice signal processing and failure of tone-shifting.

Regarding the errors in voice signal processing, the detection accuracy of frames in which utterances start is about 88.94%. The detection accuracy is the percentage of hummings whose actual number of utterances matches those extracted by voice signal processing in the range of ± 2. Moreover, more errors can be included through the pitch tracking process. Thus, the retrieval accuracy for the present SoundCompass seems to be less than 90%. We can not evaluate the retrieval accuracy of a query-by-humming system apart from the accuracy of voice signal processing. In addition, as the number of users increase, the signal processing errors increase, because the voice variation and the ways of singing increase. As a result, there will be voice signals that can not be handled with our current voice signal procedures. Thus, we have to investigate the hummings that are not retrieved to improve the voice signal procedures. The upper limit of accuracy can be found by repeating experiments with many users.

Regarding tone-shifting, we tried to use the highest tone as the base-tone as well as the average tones. As a result, there are no differences in retrieval accuracy between the two cases. Thus, we need to conduct more investigations to address this problem.

7. CONCLUDING REMARKS

In this paper, we described techniques to time-normalize and generate sub/subsubsequences for music data (time-scalable and tone-shiftable time-series data) for a query-by-humming system called SoundCompass. These techniques realize fast and accurate query-by-humming, since linear matching with both pitch and duration of notes becomes possible. Moreover, feature vectors with variable slide-length and AND/OR combination matching of feature vectors were proposed to increase the retrieval accuracy. Our latest version of SoundCompass incorporates all these techniques, and their effectiveness was quantitatively evaluated.

Acknowledgments

Authors would like to thank Gengo Suzuki, Makoto Onizuka, Yoshiaki Ito, and Kiyoshi Igarashi for their great help and also all people who gladly hummed a lot of tunes for us.

8. REFERENCES

[1] W. P. Birmingham, R. B. Dannenberg, G. H. Wakefield, M. Bartsch, D. Bykowski, D. Mazzoni, C. Meek, M. Mellody, and W. Rand. MUSART: Music Retrieval Via Aural Queries. In *Third International Conference on Music Information Retrieval*, 2001.

[2] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast Subsequence Matching in Time-Series Database. In *Proceedings of the ACM SIGMOD, International Conference on management of Data*, pages 419–429, 1994.

[3] J. Foote. Visualizing Music and Audio using Self-Similarity. In *Proc. ACM Multimedia 99*, pages 77–80, November 1999.

[4] A. Ghias, J. Logan, and D. Chamberlin. Query By Humming. In *Proc. ACM Multimedia 95*, pages 231–236, November 1995.

[5] J.-S. R. Jang and H.-R. Lee. Hierarchical Filtering Method for Content-based Music Retrieval via Acoustic Input. In *Proc. of the 9th ACM International Conference on Multimedia*, pages 401–410, 2001.

[6] E. Keogh. Exact Indexing of Dynamic Time Warping. In *Proc. of the 28th VLDB Conference*, 2002.

[7] N. Kosugi, H. Nagata, and T. Nakanishi. Query-by-Humming on Internet. In *14th DEXA 2003 Proceedings*, pages 589–600, 2003.

[8] N. Kosugi, Y. Nishihara, T. Sakata, M. Yamamuro, and K. Kushima. A Practical Query-By-Humming System for a Large Music Database. In *Proc. of the 8th ACM International Conference on Multimedia*, pages 333–342, 2000.

[9] Rodger McNab. INTERACTIVE APPLICATIONS OF MUSIC TRANSCRIPTION. Master's thesis, Computer Science at the University of Waikato, 1996.

[10] T. Nishimura, H. Hashiguchi, J. Takita, J. Xin Zhang, M. Goto, and R. Oka. Music Signal Spotting retrieval by a Humming Query Using Start Frame Feature Dependent Continuous Dynamic Programming. In *Third International Conference on Music Information Retrieval*, 2001.

[11] L. Rabiner and B.-H. Juang. *FUNDAMENTALS OF SPEECH RECOGNITION*. PTR Prentice-Hall, Inc, 1993.

[12] Y. Zhu and D. Shasha. Warping Indexes with Envelope Transforms for Query by Humming. In *Proceedings of the ACM SIGMOD, International Conference on management of Data*, pages 181–192, 2003.