AUDIO-TO-MIDI SIMILARITY FOR MUSIC RETRIEVAL

Fábio Goródscy
University of São Paulo
fabiog@ime.usp.br

Shayenne Moura
University of São Paulo
shayenne@ime.usp.br

Marcelo Queiroz
University of São Paulo
mqz@ime.usp.br

ABSTRACT

Finding similar aspects in sound recordings is a great concern in automated music analysis applications. In this paper, we are seeking to measure how well can state-of-the-art melody-extraction algorithms be used in creating abstractions of single-voiced audio recordings for querying. Having as motivation the development of a query-by-humming application, we created experiments where we compared the performance of features automatically extracted from wav recordings, along with its ground-truth directly calculated from MIDI files. Lastly, we discuss results showing that pitch interval representations ignoring time information can preserve some discrimination capability. The aim is to explore the limits of less informative data representations that may help deciding between comparing audio-to-MIDI or audio-to-audio in query-by-humming applications.

1. INTRODUCTION

Retrieving music from a dataset is a common task nowadays. People listen to music and they usually want to remember names of songs that linger on their memory. However, much time and effort may be spent to find one specific version of a song. Frequently, a user is unable to remember the lyrics, the artist or any other common meta-data from the music she/he is looking for; in such cases the only resource available is to hum the melody that is present in their memories.

This yields an important problem in Music Information Retrieval: to search within a dataset using only hummed/sung queries. It is necessary to transform the record into a representation that could be matched with the stored data in order to retrieve efficiently the correct music to the user.

Recovering information based on sung queries has two main challenges: codification of the information, and similarity criteria. There are works as *Tararira* [1] that focus on the note's pitch and duration to construct the codification and use this information to find similar items in a dataset. One algorithm that uses this kind of symbolic information is the *SMBGT* [2], a subsequence matching framework that allows gaps on queries and targets, where you can control parameters (variance tolerance levels, maximum match length and minimum number of matched elements) to improve the retrieval.

Another approach is to use the fundamental frequency line of melody to match with the target's melodic line. In the *Follow That Tune* system [3], a modified DTW has been used in order to calculate the alignment between query and targets. Using a modified representation to summarize melodic information, Salamon [4] has built a retrieval algorithm using the Q_{max} algorithm to compute fitness values and rank targets in a dataset (by sorting them in decreasing fitness order).

This paper deals with the problem of searching hummed queries within a dataset that uses MIDI representations. For reliability of the methods and algorithms, we used different audio to MIDI transcription systems (Melodia [5] and ASyMuT [6]) and we are concerned to make the evaluation of their performances using a custom-made version of the SMBGT algorithm. In addition, we discuss the algorithm limitations using this kind of representation, comparing the human recognition of the humming record and MIDI representation.

The article is organized as follows: section 2 brings a brief introduction of relevant concepts that allows the automatic comparison of melodic information. Section 3 describes the database, the feature extraction and the matching algorithms. Section 4 describes the evaluation using two different audio to MIDI transcriptions. The results are discussed in section 5.

2. CONCEPTS

2.1 MIDI transcription

The process of automatically transcribing audio representations of hummed sounds into symbolic versions, such as MIDI, has been explored for years, and is far from being a closed problem. Nevertheless, people usually follow a similar strategy when working in this task, which we will describe in this section.

The first step for MIDI transcription is converting the audio signal into some feature space that highlights fundamental frequencies of the voiced segments in the audio signal. This is a temporal series of fundamental frequencies that can be easily converted to a pitch sequence, which is the expected output for this task. This step is less controversial and it is mostly solved for monophonic recordings, even though for polyphonic signals it is still very challenging. ASyMuT [6] is a system designed for transcribing monophonic recordings; this algorithm creates the pitch vector by analyzing the spectral representation of the

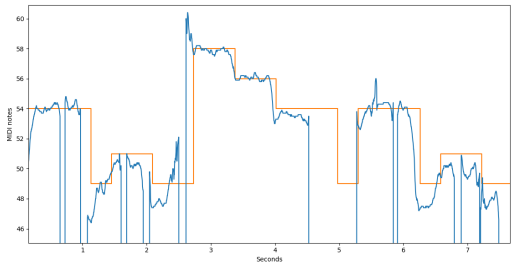


Figure 1: A pitch vector (orange) automatically extracted from a hummed query, aligned against its MIDI database version (blue).

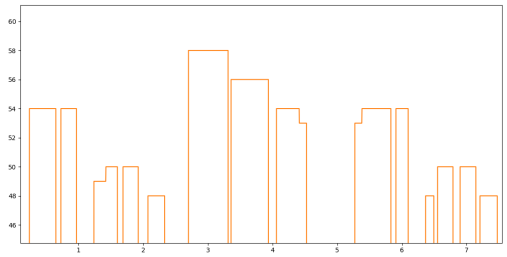


Figure 2: Final transcription after smoothing, rounding, and removing short duration notes from a pitch vector.

audio, looking for harmonic series of spectral peaks with maximal sum. Melodia [5] is designed for polyphonic signals, although it also works for monophonic signals. It has a behavior similar to ASyMuT, but it takes an extra step of evaluating the pitch candidates evolution in time, discarding extraneous pitch jumps. Figure 1 shows the results from this step.

The second step in MIDI transcription takes pitch vectors anc converts them into a sequence of discrete events, each one with specified pitch and duration, in a format similar to that of a MIDI file (that uses `note_on`/`note_off` events). This is the most challenging step, and there is no simple setting of the transcription algorithms that is always guaranteed to produce good results. A common pipeline for this task is to smooth the pitch vector, making jumps and fluctuations in instantaneous frequency less intense. This smoothing step is usually done by digital filters; however, choosing the right filter is hard and the most appropriate filter may vary for each recording, which is part of the challenge for this step. After obtaining the smoothed pitch vector, the pitch values can be rounded to the closest integer MIDI note. The last step is to find groups of MIDI notes that meet certain criteria (e.g. minimum duration). The final transcription is then similar to figure 2.

2.2 Interval representation

Absolute pitch values, or MIDI note values, are valuable for comparing recordings and for trying to find one specific record within a database. In order to introduce a degree of tolerance in the comparison of versions and melody contours, one solution is to consider the differences of consecutive pitch values, or *intervals*; intervals can lead to better contour matching by allowing tonal independence. One question that still has to be considered regards octave equivalence: frequently the representation of symbolic sequences can be simplified to allow only 12 pitch classes (integers between 0 and 11) instead of absolute pitch values, and also pitch class intervals instead of absolute intervals.

2.3 Time Series Matching

Ultimately, what we obtain is a sequence of timestamped events. There are various approaches for the alignment of two such sequences. One of these approaches is broadly used when building query-by-humming applications, the Dynamic Time Warping [7]. It is the default approach when comparing pitch vectors of dissimilar lenghts, and is one of the best approaches for query-by-humming when trying to maximize the recall and MRR of the results, according to Salamon et al. [8]. Even though DTW has several advantages, such as its retraceability [1] , it is not easy to adjust it to the interval representation, because the database intervals generated from the pure symbolic representation have a completely different structure than the intervals obtained by automatic transcription from the audio recording. Thus, we used another matching approach, the SMBGT algorithm [2], which is also a dynamic programming algorithm for matching sub-sequences.

The representation used and the matching algorithm have minor overall modifications compared to [2]. The SMBGT follows these steps: a sub-sequence A is compared element by element to a sub-sequence B, a matrix is calculated by placing 0 plus the last position of the matrix whenever a element is considered different (i.e. does not satisfy a pre-established condition) or 1 plus the last position of the matrix whenever it is considered equal. The algorithm has a parameter for resetting the cell score whenever it hops through a number of different cells without finding a equal element. The final score for the comparison of the two sequences is the highest value over the whole matrix, usually divided by the size of the smaller sequence, as this reflects the percentage of the longer equal sequence.

3. EXPERIMENT

3.1 Database

We considered the database that has been used in the MIREX annual competition (*Music Information Retrieval Evaluation eXchange*) for our experiments. It consists of 4431 WAV recordings of hummed queries, sampled at 8kHz. The queries were gathered through a period of 7 years, and include 195 participants and 48 songs (not all participants recorded every song in the set). This database of queries is processed and matched against a MIDI database that includes the same 48 songs among other 8474 MIDI songs from ESSEN folk music database. These 8474 extra songs were not hummed by the participants, thus they are used

[1] where we can find the exact sub-sequence that has been automatically matched to the input sequence

solely for the purpose of making the music retrieval task harder.

3.2 Feature extraction

The feature extraction algorithm works similarly to what has been described in section 2. All WAV queries are first processed for F0 determination within segmented audio frames. This gives an F0 temporal series that are later used for smoothing and approximating MIDI note values, and finally to create MIDI files. The MIDI file is used for finding intervals of two consecutive notes, yielding a sequence of intervals, which is the representation used in this experiment. We used two different implementations for this task, both of which are freely available in the Internet at the time of writing. The first one is based on Melodia [4] and its code can be found online [2] . The other implementation used was ASyMuT [6], and its code can also be found online [3] . The transcription steps are illustrated in figure 3.

3.3 Matching

Symbolic representations, such as those resulting from hummed transcriptions and MIDI re-encoding, reduce the complexity of the original melody. We need similarity measures that may be used to compare such simplified representations and the database MIDI files. For this task we used the SMBGT algorithm, which is a dynamic programming approach for the problem of comparing two sub-sequences, which was proposed within the context of a query-by-humming application [2].

MIDI transcriptions from each WAV file in the dataset are matched against every MIDI contained in the repertoire dataset. Therefore, each WAV receives 8522 similarity scores, from which a list of 10 top matches are kept. In this way, we expect to find not only the exact match, but other pieces with high melodic similarity compared to the hummed query. Even though it would be desirable, it is not reasonable to expect that the desired song is always in the first position of this list, because the tolerance added to the matching algorithm introduces ambiguities that result in high scores for many other melodies besides the ground-truth.

4. EVALUATION

Our evaluation proposal has been aimed at measuring how well the proposed method performs in trying to keep the ground-truth result among the 10-highest-scoring MIDI files. This would give us a hint on the system's ability to identify that a hummed melody matches part of a file in the dataset. For this purpose, we have taken the recall of the ground-truth belonging to the list of the 10-highest-scoring melodies. Additionally, we use the mean reciprocal rank (MRR) to find out how frequently the exact match appears in first position. Even though those measures might be seen as "quality" or "capability" of the system in giving the expected answer, we rather look at then as characteristics of the method. This way, the goal is not to achieve high recall and MRR measures, but to describe what kind of music result it is retrieving.

Smooth	Recall	MRR	Mean Position
0.05	4%	0.02	38.52
0.01	32%	0.23	4.35
1.0	6%	0.02	84.01

Table 1: Values for Melodia transcription.

Smooth	Recall	MRR	Mean Position
0.064	32%	0.19	5.13
0.128	13%	0.06	16.51

Table 2: Values for ASyMuT transcription.

Recall calculation is made by counting the number of times that ground-truth MIDI appears in the list of results, divided by the total number of queries in the dataset; i.e., recall will be 1 if every expected MIDI is returned by the system, 0.5 if half of them are retrieved, or 0 if no correct MIDI is found. MRR can be seen as the harmonic mean of the positions of the ground-truth in the retrieved list for each query; i.e. MRR will be equal to recall if every correct answer appears in first position, half the MRR if every answer is in the second position, and so on.

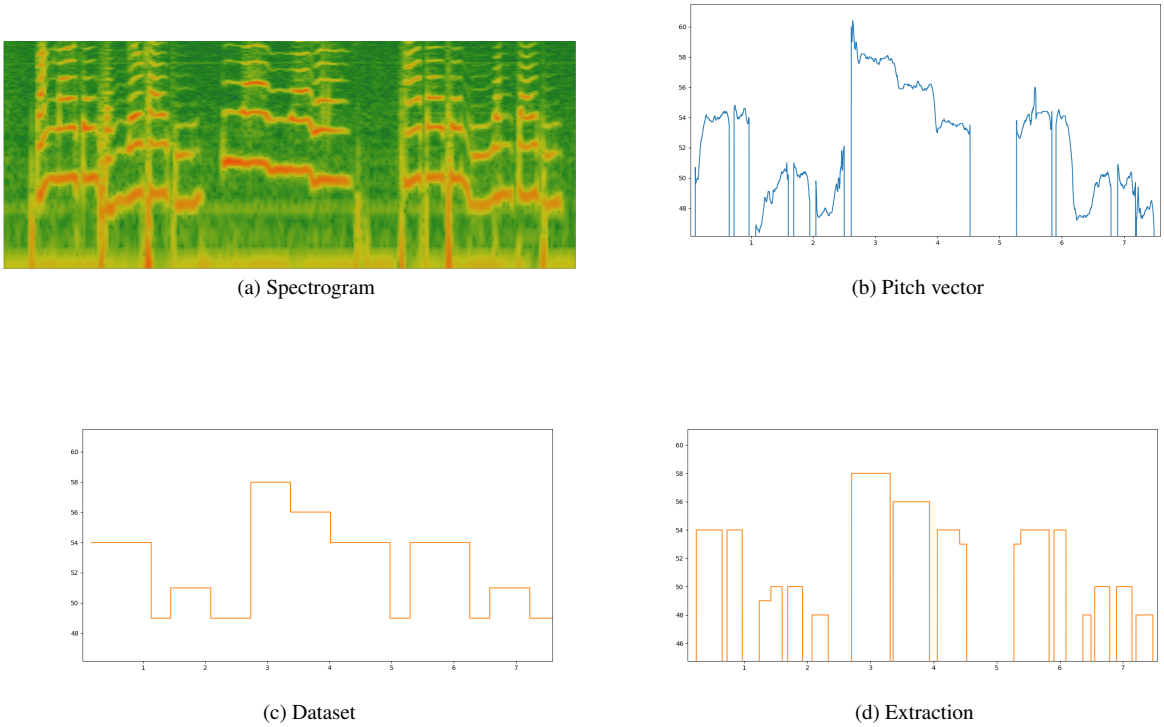
An important consideration regarding these metrics is their relationship to the quality of the transcription of the WAV files. The transcription algorithms have parameters for defining smoothing levels, which makes transcriptions more accurate or more tolerant to F0 deviations; setting these parameters to extreme values affects the transcription quality and the recall and MRR metrics, as can be seen in tables 1 (for MIDI transcriptions using Melodia) and 3 (for MIDI transcriptions from ASyMuT).

Furthermore, the choices of parameters for the matching algorithms also lead to changes in recall and MRR values. In the SMBGT algorithm we have explored the parameter controlling the different sizes of gaps allowed within subsequences. In our experiments, recall ranged from 34.5% to 37.0% for any combination of gaps, which means that optimization of this parameter could lead to only 2.5% improvement of recall, so that even though there is room for improvement, its impact is not considerable.

Another important point was to choose the representation that would be used for matching. We have tried to use rhythmic information from the automatic MIDI transcriptions. For comparison, we used as baseline a random classification, based on random MIDI similarity values irrespectively of the file contents. The algorithms that we used focused in keeping melodic information despite of rhythm information loss, which made our rhythmic tests always achieve very poor recall values, being in the range between 13% and 17%.

[2] https://github.com/justinsalamon/audio_to_midi_melodia

[3] https://github.com/adrianomitre/asymut



(a) Spectrogram

(b) Pitch vector

(c) Dataset

(d) Extraction

Figure 3: Steps of algorithmic transcription in our approach. a) Spectrogram of a hummed query. The strong lines in the bottom are captured by the F0-extraction algorithm. b) Result of F0 extraction. It has a similar shape to the strongest line of the spectrogram. c) MIDI file as seen in database. d) MIDI created from pitch vector.

Interval type	Recall	MRR	Mean Position
Regular	38%	0.23	4.29
Pitch classes	37%	0.23	4.33

Table 3: Recall and MRR values for different interval representations.

5. DISCUSSION

In an ideal world, a query-by-humming application would approach the behavior of the human cognition with respect to melodic similarity. As far as we have gone into this experiment, this idealized goal is not within reach by the methodology here described. One alarming behavior of the framework described is that it will frequently bring results that no human would classify as similar, for most of the queries. Furthermore, melodic similarity values will usually not indicate clearly what the correct answer is; there is always a high degree of confusion among competing MIDI files. Another problem of this methodology is that the transcription algorithms usually shorten sung notes, often because the moment of attack or release, or the period of transition between notes, makes the algorithm unsure of the pitch value in those moments and they tend to be discarded. But this note shortening makes it hard to define the parameter controlling the minimal duration for notes, because this choice would not be based on the actual problem (i.e. based on real data it would be good enough to set it to between 0.5s and 0.1s), but it would be influenced by the computer representation and the transcription errors introduced in the F0-extraction.

Regarding the choice of using absolute pitch values, pitch classes or intervals, it might be better to use the full absolute pitch vector, even though it incurs in a higher computation cost. Moreover, audio-to-audio alignment might also be an alternative to consider, if a matching strategy is defined to account for the fact that the query is tipically monophonic whereas the music database would be polyphonic (i.e. the hummed query should be aligned to the most prominent melody within the polyphonic texture).

6. CONCLUSIONS

In this paper, we considered the problem of retrieving a song from a dataset using MIDI representations calculated from hummed queries. We described the main concepts related to this task, particularly the transcription step and the SMBGT matching algorithm. We gave details of our experiments and discussed their results. Finally, we discussed some aspects relating this kind of representation to the human perception of melodic similarity, highlighting difficulties we have found. Lot of effort has been spent over this task, still it is not marked as completed. Furthermore, approaching human cognition with computers will always be complicated, and this task is a closely related to listening cognition.

Acknowledgments

The first two authors acknowledge the support by CNPq and SEMPRE: Society for Education, Music and Psychology Research.

7. REFERENCES

[1] E. López, M. Rocamora, and G. Sosa, "Búsqueda de música por tarareo," 2004.

[2] A. Kotsifakos, P. Papapetrou, J. Hollmen, and D. Gunopulos, "A subsequence matching with gaps-range-tolerances framework: A query-by-humming application," 2011.

[3] B. Stasiak, "Follow that tune – adaptive approach to dtw-based query-by-humming system," *ARCHIVES OF ACOUSTICS*, vol. 39, no. 4, p. 467–476, 2014.

[4] J. Salamon, E. Gómez, D. P. W. Ellis, and G. Richard, "Melody extraction from polyphonic music signals: Approaches, applications and challenges," *IEEE Signal Processing Magazine*, In Press (2013).

[5] J. Salamon and E. Gómez, "Melody extraction from polyphonic music signals using pitch contour characteristics," *IEEE TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING*, 2012.

[6] A. Mitre and M. Queiroz, "Um sistema automático de transcrição melódica," 2005.

[7] M. Müller, *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications*, 2015.

[8] J. Salamon, J. Serrà, and E. Gómez, "Tonal representations for music retrieval: from version identification to query-by-humming," *Int. J. of Multimedia Info. Retrieval, special issue on Hybrid Music Info. Retrieval*, vol. 2, no. 1, pp. 45–58, Mar. 2013. [Online]. Available: http://dx.doi.org/10.1007/s13735-012-0026-0