LARGE-SCALE COVER SONG RECOGNITION USING HASHED CHROMA LANDMARKS

Thierry Bertin-Mahieux and Daniel P. W. Ellis

LabROSA, Columbia University
1300 S. W. Mudd, 500 West 120th Street,
New York, NY 10027, USA
tb2332@columbia.edu, dpwe@ee.columbia.edu

ABSTRACT

Cover song recognition, also known as *version identification*, can only be solved by exposing the underlying tonal content of music. Apart from obvious applications in copyright enforcement, techniques for cover identification can also be used to find patterns and structure in music datasets too large for any musicologist to listen to even once. Much progress has been made on cover song recognition, but work to date has been reported on datasets of at most a few thousand songs, using algorithms that simply do not scale beyond the capacity of a small portable music player. In this paper, we consider the problem of finding covers in a database of a million songs, considering only algorithms that can deal with such data. Using a fingerprinting-inspired model, we present the first results of cover song recognition on the Million Song Dataset. The availability of industrial-scale datasets to the research community presents a new frontier for version identification, and this work is intended to be the first step toward a practical solution.

Index Terms— Cover song, fingerprinting, music identification, Million Song Dataset

1. INTRODUCTION

Making sense of large collections of music is an increasingly important issue, particularly in light of successful and profitable exploitations of archives of millions of songs by commercial operations such as Amazon and iTunes. The field of music information retrieval (MIR) comprises many tasks including fingerprinting [1] and music recommendation [2]. Fingerprinting involves identifying a specific piece of audio, whereas recommendation deals with finding "similar" songs according to some listener-relevant metric. Cover song recognition sits somewhere in the middle of these tasks: The goal is to identify a common musical work that might have been extensively transformed and reinterpreted by different musicians. Commercial motivations for cover song recognition include copyright enforcement. More interestingly, finding and understanding "valid" transformations of a musical piece can help us to develop intelligent audio algorithms that recognize common patterns among musical excerpts.

Cover song recognition has been extensively studied in recent years, but one significant shortcoming was the lack of a large-scale collection with which to evaluate different approaches. Furthermore, small datasets do not penalize slow algorithms that are impractical to scale to "Google-sized" applications. As a reference, The Echo Nest[1] claims to track approximately 25M songs. To address the lack of a large research dataset, we recently released the Million Song Dataset [3] (MSD) and a corresponding list of cover songs, the SecondHandSongs dataset[2] (SHSD). The SHSD lists $12,960$ training cover songs and $5,236$ testing cover songs that all lie within the MSD. The MSD includes audio features and various kinds of metadata for one million tracks. The audio features include chroma vectors, which is the representation most commonly used in cover song recognition.

This paper presents the first published result on the SHSD (to our knowledge) and the first cover song recognition result on such a large-scale dataset. The dataset size forces us to set aside the bulk of the published literature since most methods use a direct comparison between each pair of songs. An algorithm that scales as $O(n^2)$ in the number of songs is impractical. Instead, we look for a fingerprinting-inspired set of features that can be used as hash codes. In the Shazam fingerprinter, Wang [1] identifies landmarks in the audio signal, i.e., recognizable peaks, and records the distances between them. These "jumps" constitute a very accurate identifier of each song, robust to distortion due to encoding or from background noise. For cover songs, since the melody and the general structure (e.g. chord changes) are often preserved, we can imagine using sequences of jumps between *pitch* landmarks as hash codes.

A hashing system contains two main parts. Extraction computes the hash codes (also called fingerprints) from a piece of audio. It can be relatively slow since it is a once-only process that scales as $O(n)$ in the number of songs, and can easily be parallelized. The second part, Search, retrieves hash codes from a database given a query song. This has to be optimized so (1) hash codes are easy to compare, e.g., encoded in a single integer, and (2) the number of hash codes per song is small enough to give a tractable total database size.

Note that a full system will typically consist of multiple stages, each filtering out the majority of remaining non-covers using increasingly expensive techniques. We focus on the first layer that has to be extremely fast. On a million songs, our long-term goal is to retain only 1% of the tracks, i.e. 10K, as possible covers. This reduced set can then be processed by a second, slower and more accurate process. The following section lists such algorithms that have been reported in the literature.

TBM is supported in part by a NSERC scholarship. This work was partly funded by a grant from Google.

[1] `http://the.echonest.com`

[2] `http://labrosa.ee.columbia.edu/millionsong/secondhand`

2. RELATED WORK

The most complete introduction to cover song recognition can be found in [4]. State-of-the-art algorithms consist of the 5 steps summarized below:

1. Features extraction: Usually chroma or PCP. Some methods try to extract the melody or the chords.
2. Key invariance: A cover song can be performed in a different key. With chroma features, this means that the cover can have identical features up to a rotation on the chroma axis.
3. Tempo invariance: Using beat tracking or dynamic time warping to compute features on musically-relevant time frames, invariant in a slower or faster interpretation.
4. Structure invariance: An optional step, one can search, for instance, for repeated patterns, thus further abstracting the representation.
5. Similarity computation: Dynamic programming alignment, or some simpler distance if the structure invariance step simplified the data enough.

Later in this paper, we provide a partial comparison with the method from [5] that follows this 5-step structure.

One can see that the final alignment step has a time complexity of $O(n^2)$ where n is the number of songs in the database. This is a prohibitive step when working with the MSD. Some researchers have proposed the use of hashing, which circumvents this problem. Pioneering work include [6, 7], but none of this previous work uses datasets larger than a few thousand tracks – far smaller than the MSD.

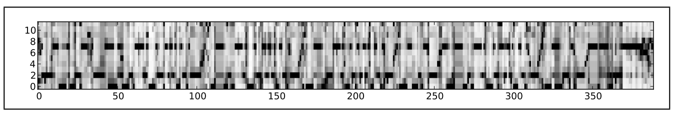

Figure 1: Beat-aligned chromas of the song "Wild Rover" by *Dropkick Murphys*.

3. HASHING SYSTEM

In this Section we present the core of the algorithm, i.e., how to move from a chroma matrix to a relatively small set of integers for each song.

3.1. Data representation

The feature analysis used throughout this work is based on Echo Nest analyze API [8]. A chromagram (Figure 1) is similar in spirit to a constant-Q spectrogram except that pitch content is folded into a single octave of 12 discrete bins, each corresponding to a particular semitone (i.e., one key on a piano). For each song in the MSD, the Echo Nest analyzer gives a chroma vector (length 12) for every music event (called "segment"), and a segmentation of the song into beats. Beats may span or subdivide segments. Averaging the per-segment chroma over beat times results in a beat-synchronous chroma feature representation similar to that used in [5]. The beat-synchronous chroma feature representation corresponds to the 3rd step from Section 2.

The most appropriate normalization for chroma vectors is not obvious. For instance, should we divide each column by its maximum value, or should we take into account segment loudness (another attribute provided by The Echo Nest analyzer)? In this work we normalize by the max, and we do not use loudness, but these choices were made empirically and may not give the best performance with other algorithms.

3.2. Hash codes

Having calculated the normalized beat-synchronous chroma matrix, we first average the chroma over pairs of successive beats, which reduced dimensionality without impacting performance in our experiments. We then identify landmarks, i.e. "prominent" bins in the resulting chroma matrix. We use an adaptive threshold as follow:

- Initialize threshold vector $T_i^0, i = 1 \ldots 12$ as the maximum in each chroma dimension over the first 10 time frames (of two beats each).
- At time t, accept a bin as a landmark if its value is above the threshold. Let v be the value of this landmark and c be its chroma bin index. Then we set $T_c^t = v$. We limit the number of landmarks per time frame to 2.
- Moving from t to $t+1$, we use a decay factor ψ to update the threshold: $T_i^{t+1} = \psi T_i^t$.

We calculate landmarks through a forward and a backward pass, and keep the intersection, i.e., landmarks identified in both passes. We then look at all possible combinations, i.e., set of jumps between landmarks, over a window of size W. To avoid non-informative duplicates, we only consider forward jumps, or in one direction within a same time frame.

Landmark pairs provide information relating to musical objects such as chords and changes in the melody line; For clarity, we refer to them as "jumpcodes": A jumpcode consists of an initial chroma value plus a list of differences in time and chroma between successive landmarks. If we had 3 landmarks at (time, chroma) positions $(200, 2)$, $(201, 7)$ and $(204, 3)$ in the beat-aligned chroma representation (and a window length W of at least 5), the jumpcode would be $\{2, ((1, 5), (3, 8))\}$. Note that for convenience, we take the jump between chroma bins modulo 12. The complete set of jumpcodes is characteristic of the song, and hopefully also of its covers. Figure 2 shows the landmark jumps for several versions of a single composition.

In this work, we do not exploit the position of the jumpcodes in the song. It might be useful to consider ordered sets of jumpcodes to identify cover songs, but this additional level of complexity would be computationally expensive. Also, covers versions frequently alter the coarse-scale structure of a composition, so global consistency is not preserved.

3.3. Encoding and Retrieval

The jumpcodes of all reference items are stored in a database to facilitate search. We use SQLite, and like any other database, it is more efficient to encode jumpcodes as a single integer. Many scheme can be devised; we use the following to allow us to easily compute a "transposed jumpcode", i.e. the jumpcode for a song rotated on its chroma axis. Such rotation is important to be able to match cover songs that are not performed in the same key.

Given a set of k (time, chroma) landmarks within a time window of size W, $(t_1 c_1), (t_2 c_2), \ldots (t_k c_k)$, we use arithmetic and delta

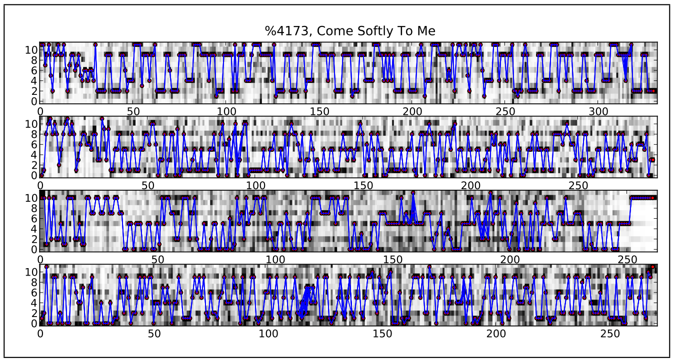

Figure 2: Landmarks and jumpcodes for the 4 cover songs of the work %4173 from the SHSD. We use the settings from "jumpcodes 2", see Table 1. In this example, the jumpcodes seem to mostly follow the melody.

coding to store these in a single value:

$$\begin{aligned} H = c_1 + 12((t_2 - t_1) + W((c_2 - c_1) \\ + 12((t_3 - t_2) + W((c_3 - c_2) \\ + \dots \\ + 12((t_k - t_{k-1}) + W((c_k - c_{k-1})))))) \end{aligned} \quad (1)$$

The most important feature of this scheme is that the initial chroma value can be retrieved via the modulo operation $\langle H \rangle_{12}$. This lets us transpose a jumpcode, i.e., construct the jumpcode we would get if the song was performed in another key, by extracting the initial chroma c_1, and adding rotation value between 0 and 11 (modulo 12) prior to recombination.

Empirically, we observe that some songs generate more jumpcodes than others (including duplicates). To avoid this biasing retrieval, we assign a weight to each jumpcode based on the number of times it appears in the song divided by the log (base 10) of the total number of jumpcodes in that song. The use of log is empirically-based, but the intuition is that it diminishes the importance of jumpcodes that appear often, if they occur in a larger pool.

For the retrieval phase, we have all jumpcodes (with positive weights) for all songs in the database. Given a query song, we get all pairs (song, jumpcode) for which (1) the jumpcode also belongs to the query song, and (2) the weight of that jumpcode falls within a margin around the weight of that jumpcode in the query song. The margin is computed given a factor α as $(1-\alpha)weight_{query} \leq weight \leq (1+\alpha)weight_{query}$. Once all the pairs from all jumpcodes in the query song have been retrieved, reference songs are ranked in likelihood of being cover versions of the query based on the number of times they appear in a matched pair.

One peculiarity of cover song recognition is that the target might be in another key. Thus, for each query, we perform 12 independent queries, one per transposition. As explained above, transposing the query's jumpcodes is computationally efficient.

4. EXPERIMENTS

After a brief description of the dataset, we present the results of our experiments.

4.1. SecondHandSong dataset

The SecondHandSongs dataset (SHSD) was created by matching the MSD with the database from `www.secondhandsongs.com` that aggregates user-submitted cover songs. It is arbitrarily divided in to training and test sets. The SHSD training set contains $12,960$ cover songs from $4,128$ cliques (musical works), and the test set has $5,236$ tracks in 726 cliques. Note that we use the term "cover" in a wide sense. For instance, two songs derived from the same original work are considered covers of one another. In the training set of the SHSD, the average number of beats per song is 451 with a standard deviation of 224.

4.2. Training

To simplify the search within algorithm parameters, we created a set of 500 binary tasks out of the $12,960$ songs from the training set. From a query, our algorithm had to choose the correct cover between two possibilities. The exact list of triplets we used are available on the project's website. We overfit this 500-queries subset through many quick experiments in order to come up with a proper setting of the parameters (although given the size of the parameter space, it is entirely possible that we missed a setting that would greatly improve results). Parameters include the normalization of beat-chroma (including loudness or not), the window length W, the maximum number of landmarks per beat, the length (in number of landmarks) of the jumpcodes, the weighting scheme of the hash codes, the allowed margin α around a jumpcode weight, the decay ψ of the threshold envelope, etc. We present the best settings we found to serve as a reference point for future researchers.

4.3. Results

We first present our results on the 500-queries subset. We report two settings for the jumpcodes. This first, called "jumpcodes 1", gave the best results on the 500-pair task and used the following parameters: $\psi = 0.96$, $W = 6$, $\alpha = 0.5$, number of landmarks per jumpcode: 1 and 4, chroma vectors aligned to each beat. Unfortunately, it generated too many jumpcodes per song to be practical. The parameters for "jumpcodes 2" were: $\psi = 0.995$, $W = 3$, $\alpha = 0.6$, number of landmarks per jumpcode: 3, chroma aligned to every 2 beats. The results are presented Table 1, where accuracy is simply the number of times the true cover was selected. Note the drop from $11,794$ to 176 jumpcodes on average per song between the two parameter sets.

As a comparison, we also report the accuracy from the published algorithm in [5], called "correlation" in the Table. This method computes the global correlation between beat-aligned chroma representations, including rotations and time offsets. We conclude that our current method performs similarly – the subset is too small to claim more than this. Since this binary task seems much easier than the 1-in-N task in [5], we expected a higher baseline; perhaps the cover songs are rather difficult in this subset, or possibly the chroma representation from The Echo Nest is less useful for cover recognition than the one in [5].

Our next experiment tested how our settings would translate to a larger set, namely the $12,960$ training cover songs. For now on we report the average rank of the target covers for each query. On the subset, using the settings from "jumpcode 2", the average position was $4,361$. If we think of a query as doing $12,960$ binary decision, this gives us an accuracy of 66.3%. We have $2,280,031$ (song,

	accuracy	#hashes
random	50.0%	-
jumpcodes 1	79.8%	11,794
jumpcodes 2	**77.4**%	176
correlation	76.6%	-

Table 1: Results on the 500-queries subset. The #hashes fields represents, on average, the number of jumpcodes we get per track.

jumpcode) pairs in the database for the training songs, giving us an average of 176 jumpcodes per track.

Finally, over the whole MillionSsong Dataset, using the SHSD test set, we get an average rank of **308, 369** with a standard deviation of **277, 697**. To our knowledge, this is the first reported result on this dataset, or any such large-scale data.

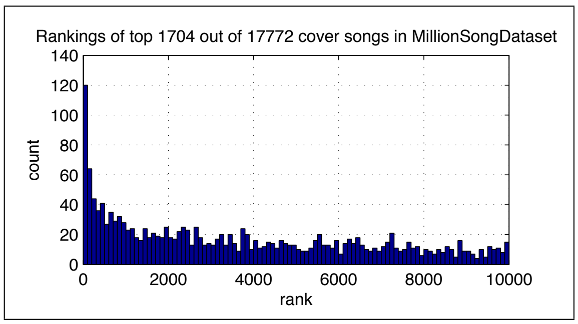

Figure 3: Histogram of the rankings of the 1,704 query pairs whose rank is below 10,000.

The large standard deviation hints that some covers are ranked much better than average. Figure 3 shows a histogram of the rankings of the 1,704 out of 17,772 queries (9.6%) whose target covers were ranked in the top 1% (10,000 tracks) of the million songs in this evaluation. The bin width is 100 rank positions. We see that 120 tracks (0.68% of queries) ranked at 100th or better (top 0.01%); a random baseline would be expected to place fewer than 2 of the 17,772 queries at this level. In fact, 12 query tracks ranked their matches at #1 in a million tracks. Some of these turned out to be actual duplicates, an existing problem of the MSD[3]. But most are genuine covers, such as the query "Wuthering Heights" by *Kate Bush* matching a version by *Hayley Westenra*, or "I don't like Mondays" by *The Boomtown Rats* retrieving a live version by *Bon Jovi* (with Bob Geldof of the Boomtown Rats performing guest vocals, recorded 16 years after the original).

4.4. Time and Implementation Considerations

Our goal was to construct a cover song detector able to search a million songs in a "reasonable time", which eventually meant less than a week on a few CPUs. Computing the jumpcodes for the entire MSD took about 3 days using 5 CPUs, or a little over 1 CPU-second per track (starting from the Echo Nest chroma features). A typical query takes less than a second per transposition to match against the full MSD. Factors influencing query time are the number of jumpcodes in the query and the number of reference songs matching each jumpcode.

Implementation-wise, we probably pushed SQLite outside its comfort zone. We could not index the ~ 150M jumpcodes in one table. We created ~ 300K tables, one per jumpcode. Since a typical query had between 1,000 and 2,000 jumpcodes (including rotations), only a small proportion of these tables were used in any single query.

[3] http://labrosa.ee.columbia.edu/millionsong/blog/11-3-15-921810-song-dataset-duplicates

5. CONCLUSION AND FUTURE WORK

We achieved our aim of providing a first benchmark result on the full SHSD / MSD collection. We have made available our code, training lists, and results so that other researchers can easily reproduce the results: see `http://www.columbia.edu/~tb2332/proj/coversongs.html`.

While the results reported here may not look as impressive as previously published numbers based on smaller sets, we emphasize that the scale of the database necessitated a much less precise approach; final accuracy could be improved by subsequent layers of search on the potential matches found in this first stage. We see this large-scale dataset as an opportunity: For cover song recognition, it gives us the data to learn a similarity metric instead of guessing one. For other music information retrieval tasks, many of which can be performed on the MSD, we hope that patterns learned from cover recognition can lead to better algorithms for recommendation and classification, etc. This work is a first step in that direction.

6. REFERENCES

[1] A. Wang, "An industrial strength audio search algorithm," in *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003)*, 2003.

[2] D. Eck, P. Lamere, T. Bertin-Mahieux, and S. Green, "Automatic generation of social tags for music recommendation," in *Advances in Neural Information Processing Systems 20*. Cambridge, MA: MIT Press, 2008.

[3] T. Bertin-Mahieux, D. Ellis, B. Whitman, and P. Lamere, "The million song dataset," in *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2011)*, 2011, (submitted).

[4] J. Serrà, "Identification of versions of the same musical composition by processing audio descriptions," Ph.D. dissertation, Universitat Pompeu Fabra, Barcelona, 2011.

[5] D. Ellis and G. Poliner, "Identifying cover songs with chroma features and dynamic programming beat tracking," in *Proceedings of the 2007 International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE Signal Processing Society, 2007.

[6] M. Casey and M. Slaney, "Fast recognition of remixed music audio," in *Proceedings of the 2007 International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE Signal Processing Society, 2007.

[7] S. Kim and S. Narayanan, "Dynamic chroma feature vectors with applications to cover song identification," in *Multimedia Signal Processing, 2008 IEEE 10th Workshop on*. IEEE, 2008, pp. 984–987.

[8] The Echo Nest Analyze, API, http://developer.echonest.com.