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 called *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 even listen to. 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, and we only consider 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. This task has been renewed by the availability of so many tracks, and this work is intended to be the first step towards 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. The music is available, and companies such as Amazon and iTunes proved that profit can arise from it. Typical music information retrieval (MIR) tasks include fingerprinting [1] and music recommendation [2]. The first one involves identifying some specific piece of audio, the second deals with finding "similar" songs according to some human metric. Cover song recognition sits somewhere in the middle. The goal is to identify a common musical work that might have been highly transformed by two different musicians. Commercial reasons for such a task include copyright infringement. More interestingly, finding and understanding human transformations of a musical piece force us to develop intelligent audio algorithms that recognize common patterns among musical excerpts.

Cover song recognition has been increasingly studied in the recent years, but one important flaw was the lack of a large-scale collection to evaluate the different published methods. Furthermore, smaller datasets do not penalize slow algorithms that could probably not scale to "Google-size" applications. As a reference, The Echo Nest[1] claims to track approximately 25M songs. To alleviate this problem, we recently released the Million Song Dataset [3] (MSD) and a complement list of cover songs, the SecondHandSongs dataset[2] (SHSD). The SHSD lists $12,960$ training cover songs and $5,236$ testing cover songs, all part of the MSD. The MSD contains audio features, and more, for one million tracks. These features include chroma vectors, the most common representation of audio for cover song recognition research.

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 most of the published literature since most methods use a direct comparison between each pair of songs. An algorithm that scales in $O(n^2)$ 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 measure the distance between them. These "jumps" constitute a very accurate identifier of each song, robust to noise due to encoding or coming from the background. For cover songs, since the melody and the general structure (e.g. chord changes) is often preserved, we can use sequences of jumps between pitch landmarks as hash codes.

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

Note that a full system could easily have two passes or more, each of these would filter out most of the tracks as non-covers. We focus on the first layer that has to be extremely fast. On a million song, our long-term goal is to retain only 1% of the tracks, i.e. 10K, as possible covers. These songs could be processed by a second, slower and more thorough process. The following section lists such algorithms that have been reported in the literature.

2. RELATED WORK

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

1. Features extraction: usually chroma or PCP. Some methods try to extract the melody or the chords.

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. Key invariance: a cover song can be performed in a different key. Using chroma features, it means that the cover can have identical features up to a rotation on the semi-tone axis.
3. Tempo invariance: using beat tracking or dynamic programming 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 repeated patterns for instance, thus abstracting the features even more.
5. Similarity computation: time warping, or some simpler distance if the structure invariance step simplified the data enough.

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

One can see that building the time-warping step, or computing repeated patterns across two songs, has a time complexity of $O(n^2)$ with n the number of songs in the database. This is a prohibitive step for a work on the MSD. Some researchers have proposed the use of hashing, which circumvents this problem. Pioneering work include [6, 7], but none of these present experiments on a dataset of a size approaching the one of the MSD (a few thousand songs at most).

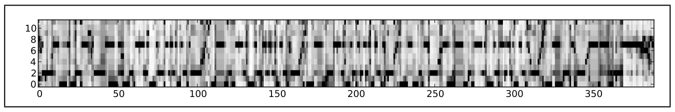

Figure 1: Beat-aligned chromas of the song "Wild Rover" by *Dropkick Murphys*.

3. HASHING SYSTEM

In this Section we present the core of the algorithm, i.e. how to pass from a chroma matrix to a few dozen 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 (e.g. 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]. Echo Nest chroma vectors are normalized to have the largest value in each column equal to 1.

The beat-synchronous chroma feature representation corresponds to the 3rd step from Section 2. There are multiple ways to renormalize or not the chroma vectors when we align them to the beats. For instance, should we divide each column by the max value, or should we take into account loudness, also 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, they might not give the best performance with all algorithms.

3.2. Hash code

We explain how to get a set of hash code from the beat-aligned chroma representation presented above.

As mentioned before, we can average the chroma values over beats. In our experiments, we saw that averaging over two beats yield similar results while reducing the dimension. We then identify landmarks, i.e. "prominent" bins in the resulting chroma matrix. We use an adaptive threshold as follow:

- Initialize T, the threshold vector of size 12 as the max per semi-tone over the first 10 time frames (one time frame = two beats).
- At time t, we accept a bin as a landmark if its value is above the threshold. Let v be the value of this landmark and T_v the threshold value for that semi-tone, we set $T_v = v$ and $T_i = \max(T_v * \psi, T_i)$, ψ being the decay factor. We also limit the number of landmarks per time frame to 2.
- Moving from t to $t+1$, we use the decay factor ψ, we get $T_{t+1} = T_t * \psi$.

We get landmarks through a forward and a backward phase and we keep the intersection, i.e. landmarks identified in both phases. Once these landmarks are identified, we look at all possible combinations, i.e. set of jumps from landmarks to landmarks, over a window of size W. To avoid non-informative duplicates, we only consider jumps forward, or in one direction if within a same time frame. The simplest set of hash codes consists of all pairs of landmarks that fall within the window length W. Such a pair can already give information about musical objects like chords, or changes in the melody line.

These sets of jumps are hash codes characteristic of the song, and hopefully also characteristic of its covers. See Figure 2 for an illustration. For sake of clarity, we refer to them as "jumpcodes". A jumpcode is a list of difference in time and semi-tone between landmarks plus an initial semi-tone (the initial time frame is always 0). If we had 3 landmarks at positions $(200, 2)$, $(201, 7)$ and $(203, 3)$ in the beat-aligned chroma representation (and a window length W of at least 4), the jumpcode would be $\{((1, 5), (2, 8)), 2\}$. Note that for convenience, we take the jump between semi-tone modulo 12.

In this work, we do not consider 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 expansive. Also, covers might use only parts of the original song, or mix the order of the song sections.

3.3. Encoding and Retrieval

The jumpcodes are entered in a database so we can use them to compare songs. We use SQLite, and like any other database, it is more efficient to encode jumpcodes as one integer. Many scheme can be devised, we use the following one that lets us easily compute a "rotated jumpcode", i.e. the jumpcode we get if we rotate the song along its semi-tone axis. Such a rotation is important, remembering that many cover songs are not performed in the same key.

We have a set of k landmarks within a time window of size W. The landmarks are pairs of (chroma bin, time frame): $(b_1, 0), (b_2, t_2), (b_k, t_k)$. The integer representation of the jumpcode can be computed along these lines:

$$\begin{aligned} & (b_1 - b_0) + (t_1 - 0 + 1) * 12 \\ + \quad & [(b_2 - b_1) + (t_2 - t_1 + 1) * 12 + 1] * 12 * (W + 1) \\ + \quad & ... \end{aligned}$$

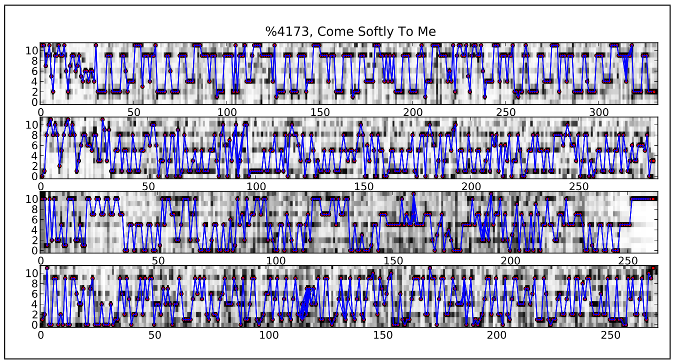

Figure 2: Landmarks and jumcodes for the 4 cover songs of the work %4173 from the SHSD. We use the settings from "jumpcodes 2", see Table 1. In this example, the jumpcodes seem to mostly follow the melody.

The initial semi-tone value can be retrieved using a modulo 12, this is the most important feature of this particular scheme. This lets us rotate a jumpcode, i.e. infer the jumpcode we would get if the song was performed in another key, by removing the initial semi-tone, and adding back a rotation value between 0 and 11.

Experiments showed us that some songs get attributed more jumpcodes than others (counting duplicates). In order to diminish that effect, we assign a weight to the jumpcodes. The weight is the number of times a particular jumpcode appears in the song divided by the logarithm (base 10) of the total number of jumpcodes in that song. Taking the logarithm is an empirical decision, but the intuition is that it diminishes the importance of jumpcodes that appear often, and put emphasis on the ones that appear only a few times.

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 percentage α as $weight_{query} * (1 - \alpha) \leq weight \leq weight_{query} * (1 + \alpha)$. Once all the pairs from all jumpcodes in the query song has been retrieved, the frequency of each song is computed. The highest the frequency, the more likely a song is a cover of the original.

One particularity of cover song recognition is that the target might be in another key. Thus, for each query, we actually do 12 independant queries, one per rotation. As explained above, the encoding scheme of the jumpcodes lets us find their rotated values in a computationally efficient way.

4. EXPERIMENT

After a brief description of the dataset, we present the results of our experiment. In the interest of making the results easier to compare and reproduce, we first report results on the training set. These results are not meant to be statistically significant, for instance we clearly overfitted the training data. That being said, a researcher experimenting on a new algorithm might be glad to compare intermediate results on such a subset as a sanity check.

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. The SHSD training set contains $12,960$ cover songs from $4,128$ cliques (musical work), the testing 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

Due to the many parameters, probably interdependent and consequently difficult to set, we created a set of 500 binary tasks out of the $12,960$ songs from the training set. From a query, our hashing 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 overfitted this 500-queries subset through many quick experiments in order to come up with a proper setting of the parameters. We do not claim we tried every combination, and it is possible that we missed a setting that greatly improves 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 so it can serve as a reference point for future researchers.

4.3. Results

We first present our results on the 500-queries subset. To be thorough, we present two settings for the jumpcodes, the first setting leading to our best result on this subset. This first one, called "jumpcodes 1" 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 gave two 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 can be seen in Table 1, accuracy is simply the number of times we selected the right cover song. Note the drop from $11,794$ to 176 jumpcodes on average per song.

As a comparison, we also report the accuracy from the published algorithm in [5], called "correlation" in the Table. This method is very intuitive, we compute the correlation between beat-aligned chroma representations, including rotations and time offsets. First we conclude that our method performs similarly. The subset is too small to claim more than that. Secondly, since this test appears easier than the one in [5], we wonder whether the cover songs are rather *difficult* in this subset, or maybe the chroma representation from The Echo Nest is less proper for cover recognition than the one use in [5] (e.g. the normalization is different).

Then, we confirm that the result we obtained on the 500 binary queries translate relatively well to a larger set, i.e. the $12,960$ training cover songs. For now on we report the average position of the covers for each queries. 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, jumpcode) pairs in the database
]

	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.

for the train songs, this gives us an average of 176 jumpcodes per track.

Finally, over the whole million song dataset using the test set of the SecondHandSongs dataset, we get an average position of $\mathbf{308,369}$ with a standard deviation of $\mathbf{277,697}$. To our knowledge, it is the first reported result on this dataset, and the first on any such large-scale data. One remark, the standard deviation is quite large, probably meaning that some covers are really faithful while others are totally different.

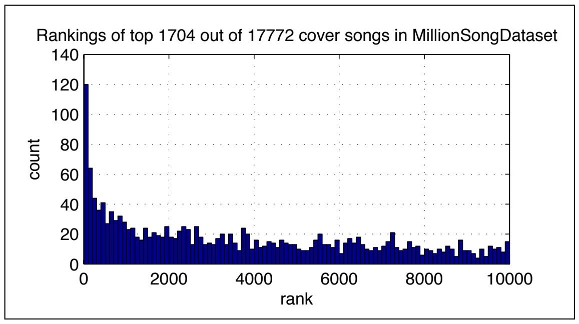

Figure 3: Histogram of the rankings of the $1,704$ query pairs whose rank is below $10,000$.

We take a closer look at the successful queries by looking at query pairs, i.e. a pair formed by a query song and a target song. Its ranking is the ranking of the target song when the dataset is queried using the query song. In Figure 3, the histogram shows the rankings of the $1,704$ out of $17,771$ query pairs (9.6%) that were ranked in the top 1% ($10,000$ tracks) of the million songs in this evaluation. We see that 120 tracks (0.68% of query pairs) ranked at 100th or better (top 0.01%); a random baseline would be expected to place fewer than 2 of the $17,771$ 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 unfortunate duplicates, an existing problem of the MSD[3]. But some are actual success, e.g. the query pair "Wuthering Heights" by (*Kate Bush*, *Hayley Westenra*) or "I don't like Mondays" by (*The Boomtown Rats*, *Bon Jovi / Bob Geldof*). This last pair is made of two different versions, both including *Bob Geldof*.

4.4. Time and Implementation Considerations

One of the requirements was for the system to work on a million song songs in a "reasonable time", mostly meaning less than a week on a few CPUs. Computing the jumpcodes took about 3 days using 5 CPUs. The $5,236$ test queries took about 4 days on 3 CPUs, RAM being a bottleneck. A typical query would take between 100 and 300 seconds. The time factors are the number of jumpcodes in the query and the number of matching songs returned for each.

Implementation-wise, we probably pushed SQLite outside its comfort zone[4]. We could not index the $\sim$ 150M jumpcodes in one table. We created $\sim$ 300K tables, one per jumpcode. Since a typical query had between $1,000$ and $2,000$ jumpcodes (rotations included), few of these tables were used in each query.

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

5. CONCLUSION AND FUTURE WORK

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

We are aware that the results reported here do not look as positive as others previously published on smaller sets, and that the task might appear less likely to be solved than before. Nevertheless, we see this 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 done on the MSD, we hope that patterns learned from cover recognition can lead to better algorithms for recommendation and classification among others. As we mentioned before, this work is meant to be 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.

[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.

[4]Lesson learned, this whole retrieval step should have been done using an efficient key/value lookup system, not a relational database. The 300K tables could even fit in RAM on a large machine!