TEMPO- AND TRANSPOSITION-INVARIANT IDENTIFICATION OF PIECE AND SCORE POSITION

Andreas Arzt[1], **Gerhard Widmer**[1,2], **Reinhard Sonnleitner**[1]
[1]Department of Computational Perception, Johannes Kepler University, Linz, Austria
[2]Austrian Research Institute for Artificial Intelligence (OFAI), Vienna, Austria
andreas.arzt@jku.at

ABSTRACT

We present an algorithm that, given a very small snippet of an audio performance and a database of musical scores, quickly identifies the piece and the position in the score. The algorithm is both tempo- and transposition-invariant. We approach the problem by extending an existing tempo-invariant symbolic fingerprinting method, replacing the absolute pitch information in the fingerprints with a relative representation. Not surprisingly, this leads to a big decrease in the discriminative power of the fingerprints. To overcome this problem, we propose an additional verification step to filter out the introduced noise. Finally, we present a simple tracking algorithm that increases the retrieval precision for longer queries. Experiments show that both modifications improve the results, and make the new algorithm usable for a wide range of applications.

1. INTRODUCTION

Efficient algorithms for content-based retrieval play an important role in many areas of music retrieval. A well known example are *audio fingerprinting* algorithms, which permit the retrieval of all audio files from the database that are (almost) *exact* replicas of a given example query (a short audio excerpt). For this task there exist efficient algorithms that are in everyday commercial use (see e.g. [4], [13]).

A related task, relevant especially in the world of classical music, is the following: given a short audio excerpt of a performance of a piece, identify both the piece (i.e. the musical score the performance is based on), and the position within the piece. For example, when presented with an audio excerpt of Vladimir Horowitz playing Chopin's Nocturne Op. 55 No. 1, the goal is to return the name and data of the piece (Nocturne Op. 55 No. 1 by Chopin) rather than identifying the exact audio recording. Hence, the database for this task does not contain audio recordings, but symbolic representations of musical scores. This is related to *version identification* (see [11] for an overview), where the goal is to identify different versions of one and the same song, mostly in order to detect cover versions in popular music.

A common way to solve this task, especially for classical music, is to use an *audio matching* algorithm (see e.g. [10]). Here, all the scores are first transformed into audio files (or a suitable in-between representation), and then aligned to the query in question, most commonly with algorithms based on dynamic programming techniques. A limitation of this approach is that relatively large queries are needed (e.g. 20 seconds), to achieve good retrieval results. Another problem is computational cost. To cope with this, in [8] clever indexing strategies were presented that greatly reduce the computation time.

In [2] an approach is presented that tries to solve the task in the symbolic domain instead. First, the query is transformed into a symbolic list of note events via an *audio transcription* algorithm. Then, a globally tempo-invariant fingerprinting method is used to query the database and identify matching positions. In this way even for queries with lengths of only a few seconds very robust retrieval results can be achieved. A downside is that this method depends on automatic music transcription, which in general is an unsolved problem. In [2] a state of the art transcription system for piano music is used, thus limiting the approach to piano music only, at least for the time being.

In addition, we identified two other limitations of this algorithm, which we tackle in this paper. First, the approach depends on the performer playing the piece in the correct key and the correct octave (i.e. in the same key and octave as it is stored in the database). In music it is quite common to transpose a piece of music according to specific circumstances, e.g. a singer preferring to sing in a specific range. Secondly, while this algorithm works very well for small queries, larger queries with local tempo changes *within* the query tend to be problematic. Of course these limitations were already discussed in the literature for other approaches, see e.g. [10] for tempo- and transposition-invariant audio matching.

In this paper we present solutions to both problems by proposing (1) a *transposition-invariant fingerprinting* method for symbolic music representations which uses an additional verification step that largely compensates for the general loss in discriminative power, and (2) a simple but effective tracking method that essentially achieves not only global, but also *local invariance to tempo changes*.

2. TEMPO-INVARIANT FINGERPRINTING

The basis of our algorithm is a fingerprinting method presented in [2] (which in turn is based on [13]) that is *invariant to the global tempo* of both the query and the entries in the database. In this section we will give a brief summary of this algorithm. Then we will show how to make it *transposition-invariant* (Section 3) and how to make it *invariant to local tempo changes* (Section 4).

2.1 Building the Score Database

In [2] a fingerprinting algorithm was introduced that is invariant to global tempo differences between the query and the scores in the database. Each score is represented as an ordered list of [ontime, pitch] pairs, which in turn are extracted from MIDI files with a suitable but constant tempo for the whole piece.

For each score, fingerprint tokens are generated and stored in a database. Tokens are created from triplets of note-on events according to some constraints to make them tempo invariant. A fixed event e is paired with the first n_1 events with a distance of at least d seconds "in the future" of e. This results in n_1 event pairs. For each of these pairs this step is repeated with the n_2 future events with a distance of at least d seconds. This finally results in $n_1 * n_2$ event triplets. In our experiments we used the values $d = 0.05$ seconds and $n_1 = n_2 = 5$ (i.e. for each event 25 tokens are created). The pair creation steps are constrained to notes which are at most 2 octaves apart.

Given such a triplet consisting of the events e_1, e_2 and e_3, the time difference $td_{1,2}$ between e_1 and e_2 and the time difference $td_{2,3}$ between e_2 and e_3 are computed. To get a tempo independent fingerprint token, the ratio of the time differences is computed: $tdr = \frac{td_{2,3}}{td_{1,2}}$. This finally leads to a fingerprint token $dbtoken = [pitch_1 : pitch_2 : pitch_3 : tdr] : pieceID : time : td_{1,2}$, with the hash key being $[pitch_1 : pitch_2 : pitch_3 : tdr]$, $pieceID$ the identifier of the piece, and $time$ the onset time of e_1. The tokens in our database are unique, i.e. we only insert the generated token if an equivalent one does not exist yet.

2.2 Querying the Database

Before querying the database, the query (an audio snippet of a performance) has to be transformed into a symbolic representation. The algorithm we use to transcribe musical note onsets from an audio signal is based on the system described in [3]. The result of this step is a possibly very noisy list of [ontime, pitch] pairs.

This list is processed in exactly the same fashion as above, resulting in a list of tokens of the form $qtoken = [qpitch_1 : qpitch_2 : qpitch_3 : qtdr] : qtime : qtd_{1,2}$. Then, all the tokens which match hash keys of the query tokens are extracted from the database (we allow a maximal deviation of the ratio of the time differences of 15%). For querying, the general idea is to find regions in the database of scores which share a continuous sequence of tokens with the query. To quickly identify these regions we use the histogram approach presented in [2] and [13]. This is a computationally inexpensive way of finding these sequences by sorting the matched tokens into a histogram with a bin width of 1 second such that peaks appear at the start points of these regions (i.e. the start point where the query matches a database position). We also included the restriction that each query token can only be sorted at most once into each bin of the histogram, effectively preventing excessively high scores for sequences of repeated patterns in a brief period of time.

The matching score for each score position is computed as the number of tokens in the respective histogram bin. In addition, we can also compute a tempo estimate, i.e. the tempo of the performance compared to the tempo in the score, by taking the mean of the ratios of $td_{1,2}$ and $qtd_{1,2}$ of the respective matching query and database tokens that were sorted in the bin in question. We will use this information for the tracking approach presented in Section 4.

3. TRANSPOSITION-INVARIANT FINGERPRINTS

3.1 General Approach

In the algorithm described above, the pitches in the hash keys are represented as absolute values. Thus, if a performer decides to transpose a piece by an arbitrary number of semi-tones, any identification attempt by the algorithm must fail.

To overcome this problem, we suggest a simple, *relative* representation of the pitch values, which makes the algorithm invariant to linear transpositions. Instead of using 3 absolute pitch values, we replace them by 2 differences, $pd_1 = pitch_2 - pitch_1$ and $pd_2 = pitch_3 - pitch_2$, resulting in a hash key $[pd_1 : pd_2 : tdr]$. For use in Section 3.2 below we additionally store $pitch_1$, the absolute pitch of the first note, in the token value.

In every other aspect the algorithm works in the same way as the purely tempo-invariant version described above. Of course this kind of transposition invariance cannot come for free as the resulting fingerprints will not be as discriminative as before. This has two important direct consequences: (1) the retrieval accuracy will suffer, and (2) for every query a lot more matching tokens are found in the database, thus the runtime for each query increases (see Section 5).

3.2 De-noising the Results: Token Verification

To compensate for the loss in discriminative power we propose an additional step before accepting a database token as a match to the query. The general idea is taken from [9] and was first used in a music context by [12]. It is based on a verification step for each returned token that looks at the context within the query and the context at the returned position the database.

Each token $dbtoken$ that was returned in response to a $qtoken$ can be used to *project the query* (i.e. the notes identified from the query audio snippet by the transcription algorithm) to the possibly matching position in the score indicated by the $dbtoken$. The intuition then is that at

true matching positions we will find a majority of the notes from the query at their expected positions in the score. This will permit us to more reliably decide if the match of hash keys is a false positive or an actual match.

To do this, we need to compute the pitch shift and the tempo difference between the query and the potential position in the database. The pitch shift is computed as the difference of the $pitch_1$ of $qtoken$ and $dbtoken$. The difference in tempo is computed as the ratio of $td_{1,2}$ of the two tokens. This information can now in turn be used to compute the expected time and pitch for each query note at the current score position hypothesis. We actually do not do this for the whole query, but only for a window of $w = 10$ notes, centred at the event e_1 of the query, and we exclude the notes e_1, e_2 and e_3 from this list (as they were already used to come up with the match in the first place).

We now take these w notes and check if they appear in the database as would be expected. In this search we are strict on the pitch value, but allow for a window of ± 100 ms with regards to the actual time in the database. If we can confirm that a certain percentage of notes from the query appears in the database as expected (in the experiments we used 0.8), we finally accept the query token as an actual match.

As this approach is computationally expensive, we actually compute the results in two steps: we first do 'normal' fingerprinting without the verification step and only keep the top 5% of the results. We then perform the verification step on these results only and recompute the scores. On our dataset this effectively more than halves the computation time.

4. PROCESSING LONGER QUERIES: MULTI-AGENT TRACKING

The fingerprinting method in [2] was mainly concerned with invariance regarding the *global tempo*. When applying this algorithm to our database with longer queries, *local tempo changes* (i.e. tempo changes within the query) prove to be problematic, because they break the 'cheap' histogram approach that is used to determine continuous regions of matching tokens.

Instead of using computationally much more expensive methods for determining these regions, we propose to split longer queries into shorter ones and track the results of these sub-queries over time. This is based on the assumption that in short queries the tempo is (quasi) stationary, and that a few exceptions will not break the tracking algorithm we use. In our implementation, we split each query into sub-queries with a window size of $w = 15$ notes and a hop size of $h = 5$ notes and then feed each sub-query to the fingerprinter individually.

Each result of a sub-query (but at most the top 100 positions that are returned) is in turn fed to an on-line position hypothesis tracking algorithm. In our current proof-of-concept implementation we use a simple on-line rule-based multi-agent approach, inspired by the beat-tracking algorithm described in [6]. For a purely off-line retrieval task a non-causal algorithm will lead to even better results.

The basic idea is to create virtual 'agents' for positions in the result sets. Each agent has a current hypothesis of the piece, the position within the piece and the tempo, and a score based on the results of the sub-queries. The agents are updated, if possible, with newly arriving data. In doing so, agents that represent positions that successively occur in result sets will accumulate higher scores than agents that represent positions that only occurred once or twice by chance, and are most probably false positives.

More precisely, we iterate over all sub-queries and perform the following steps in each iteration:

- Normalise Scores: First the scores of the positions in the result set of the sub-query are normalised by dividing them by their median. This makes sure that each iteration has approximately the same influence on the tracking process.

- Update Agents: For every agent, we look for a matching position in the result set of the sub-query (i.e. a position that approximately fits the extrapolated position of the agent, given the old position, the tempo, and the elapsed time). The position, the tempo and the score of the agent are updated with the new data from the matching result of the sub-query. If we do not find a matching position in the result set, we update the agent with a score of 0, and the extrapolated position is taken as the new hypothesis. If a matching position is found, the accumulated score is updated in a fashion such that scores from further in the past have a smaller impact than more recent ones. Each agent has a ring buffer s of size 50, in which the scores of the individual sub-queries are being stored. The accumulated score of the agent is then calculated as $score_{acc} = \sum_{i=1}^{50} \frac{s_i}{1+\log i}$, where s_1 is the most recent score.

- Create Agents: Each sub-query result that was not used to update an existing agent is used to initialise a new agent at the respective score position (i.e. in the first iteration up to 100 agents are created).

- Remove obsolete Agents: Finally, agents with low scores are removed. In our implementation we simply remove agents that are older then 10 iterations and are not part of the current top 25 agents.

At each point in time the agents are ordered by $score_{acc}$ and can be seen as hypotheses about the current position in the database of pieces. Thus, in the case of a single long query, the agents with the highest accumulated scores are returned in the end. In an on-line scenario, where an audio stream is constantly being monitored by the fingerprinting system, the current top hypotheses can be returned after each performed update (i.e. after each processed sub-query).

5. EVALUATION

5.1 Dataset Description

For the evaluation of the proposed algorithms a ground truth is needed. We need exact alignments of performances (recordings) of classical music to their respective scores such that we know exactly when each note given in the score is actually played in the performance. This data can either be generated by a computer program or by extensive manual annotation but both ways are prone to errors.

Luckily, we have access to two unique datasets where professional pianists played performances on a computer-controlled piano [1] and thus every action (e.g. key presses, pedal movements) was recorded. The first dataset (see [14]) consists of performances of the first movements of 13 Mozart piano sonatas by Roland Batik. The second, much larger, dataset consists of nearly the complete solo piano works by Chopin performed by Nikita Magaloff [7]. For the latter set we do not have the original audio files and thus replayed the symbolic performance data on a Yamaha N2 hybrid piano and recorded the resulting performances.

As we have both symbolic and audio information about the performances, we know the exact timing of each played note in the audio files. To build the score database we converted the sheet music to MIDI files with a constant tempo such that the overall duration of the file is similar to a 'normal' performance of the piece.

In addition to these two datasets the score database includes the complete Beethoven piano sonatas, two symphonies by Beethoven, and various other piano pieces. To this data we have no ground truth, but this is irrelevant since we do not actively query for them with performance data in our evaluation runs. See Table 1 for an overview of the complete dataset.

5.2 Results

For the evaluation we follow the procedure from [2]. A score position X is considered correct if it marks the beginning (+/- 1.5 seconds) of a score section that is identical in note content, over a time span the length of the query (but at least 20 notes), to the note content of the 'real' score situation corresponding to the audio segment that the system was just listening to. We can establish this as we have the correct alignment between performance time and score positions — our *ground truth*). This complex definition is necessary because musical pieces may contain repeated sections or phrases, and it is impossible for the system (or anyone else, for that matter) to guess the 'true' one out of a set of identical passages matching the current performance snippet, given just that performance snippet as input. We acknowledge that a measurement of musical time in a score in terms of seconds is rather unusual. But as the MIDI tempos in our database generally are set in a meaningful way, this seemed the best decision to make errors comparable over different pieces, with different time signatures – it would not be very meaningful to, e.g. compare errors in bars or beats over different pieces.

[1] Bösendorfer SE 290

We tested the algorithms with different query lengths: 10, 15, 20 and 25 notes (automatically transcribed from the audio query). For each of the query lengths, we generated 2500 queries by picking random points in the performances of our test database, and used them as input for the proposed algorithms. Duplicate retrieval results (i.e. positions that have the exact same note content; also, duplicate piece IDs for the experiments on piece-level) are removed from the result set.

Table 2 shows the results of the original tempo-invariant (but not pitch-invariant) algorithm on our dataset. Here, we present results for two categories: correctly identified pieces, and correctly identified piece and position in the score. For both categories we give the percentage of correct results at rank 1, and the mean reciprocal rank. This experiment basically confirms the results that were reported in [2] on a larger database (more than twice as large), for which a slight drop in performance is expected.

In addition, for the experiments with the transposition-invariant fingerprinting method, we transposed each score randomly by between -11 and +11 semitones – although strictly speaking this was not necessary, as the transposition-invariant algorithm returns *exactly* the same (large) set of tokens for un-transposed and transposed queries or scores.

Table 3 gives the results of the transposition-invariant method on these queries, both without (left) and with the verification step (right). As expected, the use of pitch-invariant fingerprints without additional verification causes a big decrease in retrieval precision (compare left half of Table 3 with Table 2). Furthermore, the loss in discriminative power of the fingerprint tokens also results in an increased number of tokens returned for every query, which has a direct influence on the runtime of the algorithm (last row in Table 3). The proposed verification step solves the precision problem, at least to some extent, and in our opinion makes the approach usable. Of course this does not come for free, as the runtime increases slightly.

We also tried to use the verification step with the original tempo-invariant algorithm but were not able to improve on the retrieval results. At least on our test data the tempo-invariant fingerprints are discriminative enough to mostly avoid false positives.

Finally, Table 4 gives the results on slightly longer queries for both the original tempo-invariant and the new tempo- and transposition-invariant algorithm. As can be seen, for the detection of the exact position in the score, using no tracking, the results based on queries with length 100 notes are worse than those for queries with only 50 notes, i.e. more information leads to worse results. This is caused by local tempo changes within the query, which break the histogram approach for finding sequences of matching tokens.

As shown on the right hand side for both fingerprinting types in Table 4, the approach of splitting longer queries into shorter ones and tracking the results takes care of this problem. Please note that for the tracking approach we check if the position hypotheses after the last tracking step match the correct position in the score. Thus, as this is an

	Score Database		Testset	
Data Description	Number of Pieces	Notes in Score	Notes in Performance	Performance Duration
Chopin Corpus	154	325,263	326,501	9:38:36
Mozart Corpus	13	42,049	42,095	1:23:56
Additional Pieces	159	574,926	–	–
Total	326	942,238		

Table 1. Database and Testset Overview. In the database, all the pieces are included. As we only have performances aligned to the scores for the Chopin and the Mozart corpus, only these are included in the test set to query the database.

Query Length in Notes	10	15	20	25
Correct Piece as Top Match	0.6	0.82	0.88	**0.91**
Correct Piece Mean Reciprocal Rank (MRR)	0.68	0.86	0.91	**0.93**
Correct Position as Top Match	0.53	0.72	0.77	**0.79**
Correct Position Mean Reciprocal Rank (MRR)	0.60	0.79	0.83	**0.85**
Mean Query Length in Seconds	1.47	2.26	3.16	3.82
Mean Query Execution Time in Seconds	0.02	0.06	0.11	0.16

Table 2. Results for different query sizes of the original *tempo-invariant* piece and score position identification algorithm on the test database at the piece level (upper half) and on the score position level (lower half). Each estimate is based on 2500 random audio queries. For both categories the percentage of correct detections at rank 1 and the mean reciprocal rank (MRR) are given. Additionally, the mean length of the query in seconds and the mean execution time for a query is shown.

	Without Verification				With Verification			
Query Length in Notes	10	15	20	25	10	15	20	25
Correct Piece as Top Match	0.30	0.40	**0.41**	0.40	0.43	0.63	0.71	**0.75**
Correct Piece MRR	0.36	0.47	**0.50**	0.49	0.49	0.69	0.76	**0.79**
Correct Position as Top Match	0.23	**0.33**	0.32	0.32	0.33	0.51	0.57	**0.60**
Correct Position MRR	0.29	0.40	**0.41**	0.40	0.41	0.59	0.66	**0.69**
Mean Query Length in Seconds	1.47	2.26	3.16	3.82	1.47	2.26	3.16	3.82
Mean Query Execution Time in Seconds	0.10	0.32	0.62	0.91	0.12	0.38	0.72	1.09

Table 3. Results for different query sizes of the proposed *tempo- and transposition-invariant* piece and score position identification algorithm on the test database with (right) and without (left) the proposed verification step. Each estimate is based on 2500 random audio queries. The upper half shows recognition results on the piece level, the lower half on the score position level. For both categories the percentage of correct detections at rank 1 and the mean reciprocal rank (MRR) are given. Additionally, the mean length of the query in seconds and the mean execution time for a query is shown.

	Tempo-invariant				Tempo- and Pitch-invariant			
	No Tracking		Tracking		No Tracking		Tracking	
Query Length in Notes	50	100	50	100	50	100	50	100
Correct Piece as Top Match	0.95	0.96	0.98	**1**	0.81	0.79	0.92	**0.98**
Correct Piece MRR	0.97	0.98	0.99	**1**	0.85	0.82	0.94	**0.99**
Correct Position as Top Match	0.78	0.73	0.87	**0.88**	0.64	0.59	0.77	**0.83**
Correct Position MRR	0.85	0.81	0.89	**0.90**	0.72	0.66	0.82	**0.86**
Mean Query Length in Seconds	7.62	15.03	7.62	15.03	7.62	15.03	7.62	15.03
Mean Query Execution Time in Seconds	0.42	0.92	0.49	1.08	2.71	6.11	3.21	7.09

Table 4. Results of the proposed tracking algorithm on the test database for both the original *tempo-invariant algorithm* (left) and the new *tempo- and transposition-invariant approach* (right), including the verification step. For the category 'No Tracking', the query was fed directly to the fingerprinting algorithm. For 'Tracking', the queries were split into sub-queries with a window size of 15 notes and a hop size of 5 notes, and the individual results were tracked by our proof-of-concept *multi-agent* approach. Evaluation of the tracking approach is based on the finding the endpoint of a query (see text). Each estimate is based on 2500 random audio queries. The upper half shows recognition results on the piece level, the lower half on the score position level. For both categories the percentage of correct detections at rank 1 and the mean reciprocal rank (MRR) are given. Additionally, the mean length of the query in seconds and the mean execution time for a query is shown.

on-line algorithm, we are not interested in the start position of the query in the score, but in the endpoint, i.e. if the query was tracked successfully, and the correct *current* position is returned. Even the causal approach leads to a high percentage of correct results with both the original and the tempo- and pitch-invariant fingerprinting algorithm. Most of the remaining mistakes happen because (very) similar parts within one and the same piece are confused.

6. CONCLUSIONS

6.1 Applications

The proposed algorithm is useful in a wide range of applications. As a retrieval algorithm it enables fast and robust (inter- and intra-document) searching and browsing in large collections of musical scores and corresponding performances. Furthermore, we believe that the algorithm is not limited to retrieval tasks in classical music, but may be of use for cover version identification in general, and possibly many other tasks. For example, it was already successfully applied in the field of symbolic music processing to find repeating motifs and sections in complex musical scores [5].

Currently, the algorithm is mainly used in an on-line scenario (see [1]). In connection with a score following algorithm it can act as a 'piano music companion'. The system is able to recognise arbitrary pieces of classical piano music, identify the position in the score and track the progress of the performer. This enables a wide range of applications for musicians and for consumers of classical music.

6.2 Future Work

In its current state the algorithm is able to recognise the correct piece and the score position even for very short queries of piano music. It is invariant to both tempo differences and transpositions and can be used in on-line contexts (i.e. to monitor audio streams and at any time report what it is listening to) and as an off-line retrieval algorithm. The main direction for future work is to lift the restriction to piano music and make it applicable to all kinds of classical music, even orchestral music. The limiting component at the moment is the transcription algorithm, which is only trained on piano sounds.

7. ACKNOWLEDGMENTS

This research is supported by the Austrian Science Fund (FWF) under project number Z159 and the EU FP7 Project PHENICX (grant no. 601166).

8. REFERENCES

[1] A. Arzt, S. Böck, S. Flossmann, H. Frostel, M. Gasser, and G. Widmer. The complete classical music companion v0. 9. In *Proceedings of the 53rd AES Conference on Semantic Audio*, 2014.

[2] A. Arzt, S. Böck, and G. Widmer. Fast identification of piece and score position via symbolic fingerprinting. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, 2012.

[3] S. Böck and M. Schedl. Polyphonic piano note transcription with recurrent neural networks. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2012.

[4] P. Cano, E. Batlle, T. Kalker, and J. Haitsma. A review of algorithms for audio fingerprinting. In *Proceedings of the IEEE International Workshop on Multimedia Signal Processing (MMSP)*, 2002.

[5] T. Collins, A. Arzt, S. Flossmann, and G. Widmer. Siarct-cfp: Improving precision and the discovery of inexact musical patterns in point-set representations. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, 2013.

[6] S. Dixon. Automatic extraction of tempo and beat from expressive performances. *Journal of New Music Research*, 30(1):39–58, 2001.

[7] S. Flossmann, W. Goebl, M. Grachten, B. Niedermayer, and G. Widmer. The Magaloff project: An interim report. *Journal of New Music Research*, 39(4):363–377, 2010.

[8] F. Kurth and M. Müller. Efficient index-based audio matching. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):382–395, 2008.

[9] D. Lang, D. W. Hogg, K. Mierle, M. Blanton, and S. Roweis. Astrometry. net: Blind astrometric calibration of arbitrary astronomical images. *The Astronomical Journal*, 139(5):1782, 2010.

[10] M. Müller, F. Kurth, and M. Clausen. Audio matching via chroma-based statistical features. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, 2005.

[11] J. Serra, E. Gómez, and P. Herrera. Audio cover song identification and similarity: background, approaches, evaluation, and beyond. In Z. W. Ras and A. A. Wieczorkowska, editors, *Advances in Music Information Retrieval*, pages 307–332. Springer, 2010.

[12] R. Sonnleitner and G. Widmer. Quad-based audio fingerprinting robust to time and frequency scaling. In *Proceedings of the International Conference on Digital Audio Effects*, 2014.

[13] A. Wang. An industrial strength audio search algorithm. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, 2003.

[14] G. Widmer. Discovering simple rules in complex data: A meta-learning algorithm and some surprising musical discoveries. *Artificial Intelligence*, 146(2):129–148, 2003.