ECAI 2012
Luc De Raedt et al. (Eds.)
© 2012 The Author(s).
This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License.
doi:10.3233/978-1-61499-098-7-67

Towards a Complete Classical Music Companion

Andreas Arzt$^{(1)}$, Gerhard Widmer$^{(1,2)}$, Sebastian Böck$^{(1)}$, Reinhard Sonnleitner$^{(1)}$ and Harald Frostel$^{(1)}$[1]

Abstract. We present a system that listens to music on-line and almost instantly identifies the piece the performers are playing and the exact position in the musical score. This is achieved via a combination of a state-of-the-art audio-to-note transcription algorithm and a novel symbolic fingerprinting method. The speed and precision of the system are evaluated in systematic experiments with a large corpus of classical music recordings. The results indicate extremely fast and accurate recognition performance — a level of performance, in fact, that even human experts in classical music will find hard to match.

1 INTRODUCTION

In this paper we describe another big step in a long-term endeavour that aims at building a musical system that is able to recognize arbitrary pieces (of classcial music, for the time being) by real-time listening, to identify the piece and provide meta-information almost instantly, and to track the performance and display the musical score in real time along with the performance. We call this, somewhat immodestly, the *Complete Classical Music Companion*.[2]

The first building block of that system – a highly robust and reactive score follower that tracks live performances and aligns the musical score to the performance in real time – was first described in [2]. In [1] this was extended with what we called 'anytime tracking ability' — the ability to tolerate arbitrary jumps, insertions, repeats, re-starts etc. on the part of the music performers. In effect, this permits the musicians to jump around in a piece in arbitrary ways — for instance, in a practicing situation — while still being correctly followed.

In the present paper, we now describe the next (and, from our point of view, penultimate) step towards building the complete classical music companion: the ability to almost instantly recognize an arbitrary piece when hearing only a few arbitrarily chosen seconds of music being played (possibly live) — the way the ideal human encyclopaedic classical music expert would. Note that the input to our system is audio streams, not some symbolic music representation such as, e.g., MIDI.

In the following, we describe the two new components that in conjunction make this possible, and the methods behind them: a real-time audio-to-pitch transcription algorithm (note recognizer), and an extremely effective and robust indexing algorithm that quickly finds matching situations in a large database of musical scores, based on partly faulty information from the note transcriber, and in the presence of possibly large differences and fluctuations in tempo and timing (which are common in classical music). We focus on a detailed experimental analysis of these two new components that together make up what might be called the *instant piece recognition ability*.

The ultimate step, not described here, is the integration of this instant recognition ability into our score follower, such that the instant recognizer constantly informs the music tracker about the most likely position and/or piece the performers might be playing at any given point in time, and in this way helps the music tracker to re-direct its focus. The resulting system will be useful for a variety of musical purposes — from fully automatic display of sheet music during practicing sessions, to real-time synchronisation of events and visualisation with live music on stage, to a comprehensive music information companion that 'knows' all of classical music and provides useful meta-information (including the score) instantly, whenever it 'hears' music.

2 THE TASK: INSTANT PIECE RECOGNITION FROM LIVE AUDIO STREAMS

As noted above, the larger context of this work is a system that listens to music (live performances) via a microphone and follows the musicians' position in the printed score (see Figure 1 for a sketch of the current system). Live input enters the system in the form of a continuous audio stream (left-hand side of Fig. 1). This audio stream is aligned, in real time, to a representation of the printed score of the corresponding piece — in our case, this score representation is another audio file that is generated from the score via some software synthesiser. Score following thus becomes an online audio-to-audio alignment problem, which is solved via a highly efficient and robust algorithm based on On-line Dynamic Time Warping, with some specific enhancements (see [2]). Figure 1 indicates that there are multiple trackers simultaneously considering and tracking different alternative hypotheses within a piece (e.g., the performers obeying a repeat sign, or ignoring it).

The task of the new *Instant Piece Recognition* function is to immediately recognize, from just a few seconds of live audio, what piece is currently being played, and exactly which passage within the piece, and to inform the trackers accordingly. This would permit musicians to start playing an arbitrary piece, at an arbitrary position, at any time, without having to give any directions to the system. The recognition process involves analysing the last few seconds of audio and searching in the score database for note configurations that match what is being 'heard'. As mentioned above, we will decompose this into two separate problems (shown as yellow boxes in the Figure 1): note recognition (transcription) from the audio stream, and search for possibly matching musical situations in the score database (denoted as symbolic music matching in the figure). Both problems are nontrivial. Automatic audio transcription is still a wide open research field (see e.g., [3, 4]), and nothing close to 100% recognition accu-

[1] $^{(1)}$ Department of Computational Perception, Johannes Kepler University Linz, Austria; $^{(2)}$ Austrian Research Institute for Artificial Intelligence, Vienna, Austria

[2] In its current state, as described here, our system knows the complete works for solo piano by Frederic Chopin (which is pretty much the complete Chopin), parts of Mozart's piano sonatas, and quite some other pieces as well.

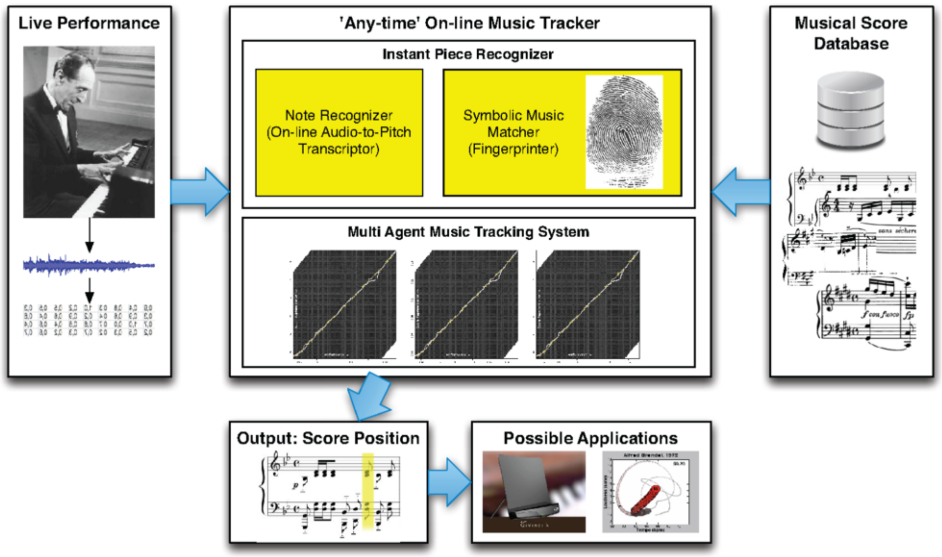
Live Performance
'Any-time' On-line Music Tracker
Instant Piece Recognizer
Note Recognizer (On-line Audio-to-Pitch Transcriptor)
Symbolic Music Matcher (Fingerprinter)
Multi Agent Music Tracking System
Musical Score Database
Output: Score Position
Possible Applications

Figure 1. Any-time Music Tracker

racy can be expected (see Table 1 below). Likewise, identifying the correct score position from imprecise and incomplete information about possibly played notes, in a large score database, and doing so in a fraction of a second, is a demanding task.

Before describing in detail our solution to these problems, we need to point out that the problem we address here is distinct from *audio fingerprinting*, which can be considered a solved problem and is in everyday commercial use. In audio fingerprinting (e.g., [6, 8]), the task is to identify a specific audio recording from an arbitrary excerpt of this same recording, possibly corrupted by noise. In other words, an audio fingerprinter can only identify recordings already in its database. Our system needs to be able to recognize a completely new rendition of a piece, for instance, a live performance currently happening on stage that has never been realized in this way before, possibly even on other instruments than any existing recordings; and the database that is being matched against contains not recordings, but symbolic music scores, in the specific form described in Section 4.1 below.

Besides audio fingerprinting the problem may also be solved via *audio matching* (i.e., the database in this case again does not consist of symbolic score representations, but of audio renditions), which in general is able to identify different recordings of the same piece. In [7] a fast method based on audio matching and indexing-techniques is proposed which is designed for off-line retrieval tasks with query lengths in the range of 10 to 20 seconds. The problem with this approach in our live setting is that we need matching results much quicker (e.g., with query sizes of about 1 second) which in our experience is not possible via a method based on audio matching techniques.

Thus to overcome the deficiencies of the existing approaches we will examine a novel kind of *symbolic fingerprinting* based on audio transcription.

3 THE NOTE RECOGNIZER

The component to transcribe note onsets from an audio signal is based on the system described in [3], which exhibits state-of-the-art performance for this task. It uses a recurrent neural network to simultaneously detect the pitches and the onsets of the notes.

For its input, a discretely sampled audio signal is split into overlapping blocks before it is transferred to the frequency domain with two parallel Short-Time Fourier Transforms (STFT). Two different window lengths have been chosen to achieve both a good temporal precision and a sufficient frequency resolution for the transcription of the notes. Phase information of the resulting complex spectrogram is discarded and only the logarithm of the magnitude values is used for further processing. To reduce the dimensionality of the input vector for the neural network, the spectrogram representation is filtered with a bank of filters whose frequencies are equally spaced on a logarithmic frequency scale and are aligned according to the MIDI pitches. The attack phase of a note onset is characterized by a rise of energy, thus the first order differences of the two spectrograms are used as additional inputs to the neural network.

The neural network consists of a linear input layer with 324 units, three bidirectional fully connected recurrent hidden layers, and a regression output layer with 88 units, which directly represent the MIDI pitches. Each of the hidden layers uses 88 neurons with hyperbolic tangent activation function. The use of bidirectional hidden layers enables the system to better model the context of the notes, which show a very characteristic envelope during their decay phase.

The network is trained with supervised learning and early stopping. The network weights are initialized with random values following a Gaussian distribution with mean 0 and standard deviation 0.1. Standard gradient descent with backpropagation of the errors is used to train the network. The network was trained on a collection of 281 piano pieces recorded on various pianos, virtual and not (seven different synthesizers, an upright Yamaha Disklavier, and a Bösendorfer SE grand piano).

Table 1 shows the transcription results for the complete test set described in Section 5.1. A note is considered to have been discovered correctly if its position is detected within the detection window around the annotated ground truth position.

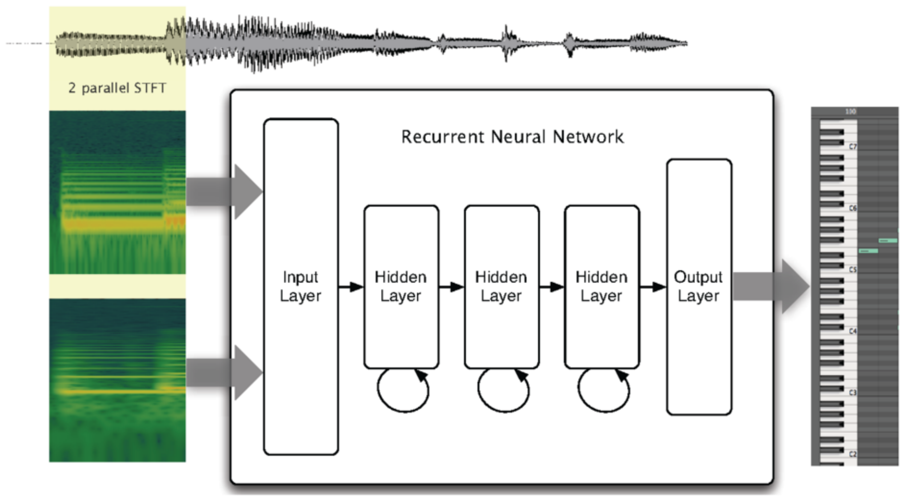

Figure 2. The Note Recognizer

Table 1. Results of the Note Transcriptor

Detection Window	Precision	Recall	F-measure
20 ms	0.585	0.486	0.531
40 ms	0.813	0.676	0.738
60 ms	0.852	0.708	0.773
80 ms	0.865	0.718	0.785
100 ms	0.870	0.723	0.790

4 THE SYMBOLIC MUSIC MATCHER

The symbolic music matcher's task is to take the output of the note recognizer and query a score database for matching positions. This is a difficult task because of two reasons. Firstly, the output of the note recognizer contains a lot of noise. As shown in table 1 only a certain percentage of the played notes is correctly recognized, and furthermore a considerable amount of wrongly detected notes is added. The symbolic music matcher needs to be robust enough to cope with this noise. Secondly, the algorithm has to deal with big differences in tempo between the score representations and the performances. Actually this manifests itself in two ways: in a global tempo difference between the query and the matching position in the score, and in local tempo deviations within the query (i.e., the performer in general does not play a constant tempo and may accelerate or slow down, while the scores given to the system are in a constant tempo without any such changes).

4.1 Building the Score Database

Before actually processing queries the score database has to be built. To do so we present the algorithm with musical scores in the format of MIDI files. In general the duration of these MIDI files is similar to the duration of a 'typical' performance of the respective piece, but without encoded timing variations. From these files a simple ordered list of note events is extracted where for each note event the exact time in seconds and the pitch as MIDI note number is stored.

Next, for each piece fingerprint tokens are generated. To make them tempo independent we create them from 3 successive events according to some constraints (also see Figure 3). Given a fixed event e we pair it 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 we then repeat this step and again pair them 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 $n_1 = n_2 = 5$.

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 we compute the time difference ratio $tdr = \frac{td_{2,3}}{td_{1,2}}$. This finally leads to a fingerprint token $[pitch_1 : pitch_2 : pitch_3 : tdr] : pieceID : time : td_{1,2}$, where the hash key $[pitch_1 : pitch_2 : pitch_3 : tdr]$ can be stored in a 32 bit integer. The purpose of storing $td_{1,2}$ in the fingerprint token will be explained in the description of the search process itself below.

The result of the score preprocessing is our score database; a container of fingerprint tokens which provides quick access to the tokens via hash keys.

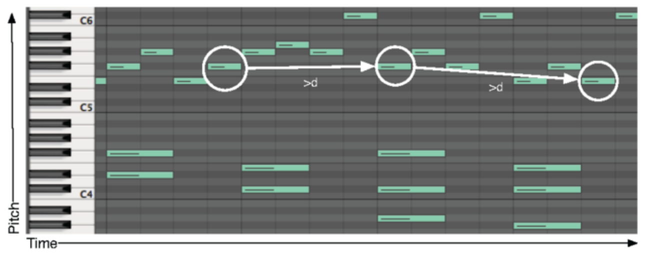

Figure 3. Fingerprint Token Generation

4.2 Querying the Database

As input the symbolic music matcher takes a list of note events with their timestamps as extracted by the note recognizer. This list is then processed in the same way as described in Section 4.1 above to produce query tokens. Of course in this case no piece ID is known and furthermore each query starts at time 0. These query fingerprint tokens are now used to query the database. The method described below is very much inspired by the audio fingerprinting method proposed in [8].

The general idea is to find regions in the score database which share a continuous sequence of tokens with the query. To do so first all the score tokens which match the query tokens are extracted from the database. When plotted as a scatter plot using their respective time stamps (see Figure 4a) matches will be indicated by (rough) diagonals (i.e., these indicate that the query tokens match the score tokens over a period of time). As identifying these diagonals directly would be computationally expensive we instead use a simpler method described in [8]. This is based on histograms (one for each piece in the score database with a time resolution of 1 second) into which the matched tokens are sorted in a way such that peaks appear at the start points of these diagonals (i.e., the start point of a query, see Figure 4b). This is achieved by computing the bin to sort the token into as the difference between the time of the score token and time of the query token. The complete process will be explained in more detail below.

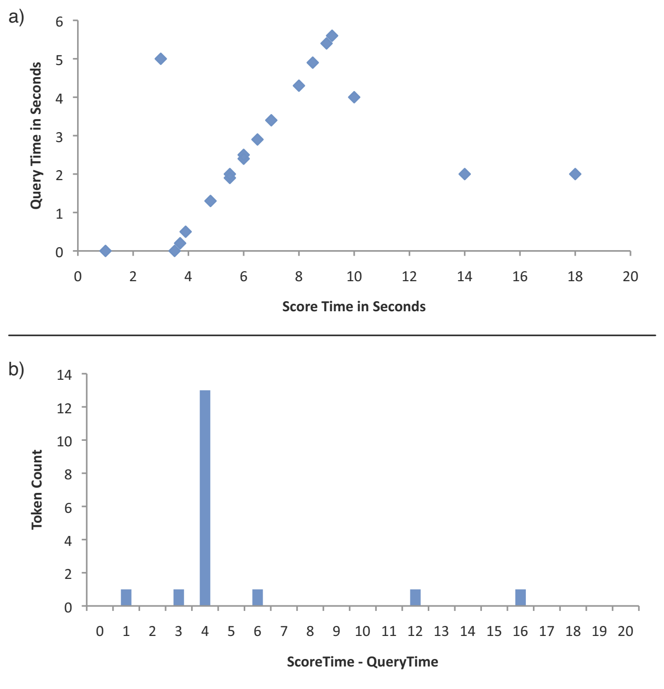

Figure 4. a) scatter plot of matching tokens and b) computed histogram for diagonal identification

For each of the query tokens qt with $[qpitch_1 : qpitch_2 : qpitch_3 : qtdr] : qtime : qtd_{1,2}$ the following process is repeated. First, matching tokens are extracted from the score database via the hash key. To allow for local tempo differences we permit the time difference ratio $stdr$ to be within $\frac{1}{4}$ of $qtdr$. This normally results in a large number of score tokens $[spitch_1 : spitch_2 : spitch_3 : stdr] : spieceID : stime : std_{1,2}$. Unfortunately directly sorting these tokens into bin $round(stime - qtime)$ of the histogram $spieceID$ does not necessarily make sense because of the query possibly having a different tempo than expected by the score.

To illustrate this let us assume a slower tempo for the query than for the respective score. Then the diagonal in Figure 4a would be steeper and when computing the bins via $round(stime - qtime)$ the first few tokens may fall into the correct bins. But soon the tokens, despite belonging to the same score position, would get sorted into lower bins instead.

Thus we first try to adapt the timing by estimating the tempo difference between the score token and the query token. First we compute the tempo ratio of both tokens $r = \frac{std_{1,2}}{qtd_{1,2}}$ and then adapt the time of the query event when computing the bin to sort the token into: $bin = round(stime - qtime * r)$.

We now have a number of histograms, one for each score in the database, and need a way of deciding on the most probable score position(s) for the query. The first method which springs to mind is to simply take the number of tokens in each bin as a score. This actually already leads to quite good results. Still this method has one problem: it favours score positions with lots of events over more sparse positions as then simply the probability to hit many tokens is higher. Thus we compute the score s of bin b as

$$s = \frac{|b|}{|query|} * \frac{|b|}{|score|}$$

In this formula $|b|$ (the number of hash tokens in bin b) and $|query|$ (the number of hash tokens in the query) are directly given. In contrast to that $|score|$ is not given as bin b only gives the starting point of the query in score, it does not make any indication about the length. It would be possible to simply assume the same tempo as in the query and count the number of tokens which are generated over the timespan of the query at this score position. Instead we compute the mean tempo of the tokens in this bin b to make an estimate of the tempo relative to the score te, estimate the length of the respective part in the score as $l = querylength * te$ and then count the number of tokens in this timespan accordingly. This proves to be a very robust way of computing the score for each bin as can be seen in the evaluation below.

5 EVALUATION

5.1 Dataset Description

For the evaluation of our algorithm a ground truth is needed, i.e. we need exact alignments of performances 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 annotation errors.

Luckily, we possess two unique datasets where professional pianists played their performances on a computer controlled piano[3] and thus every action (e.g., key presses, pedal movements) was recorded in a symbolic way. The first dataset consists of performances of the first movements of 13 Mozart sonatas by Roland Batik (described in more detail in [9]). The second, much larger, dataset consists of nearly the complete solo piano works by Chopin performed by Nikita Magaloff (see [5]). 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 performance. In addition to these two datasets we added some more scores to the database, solely to provide for more diversity and to make the task even harder for our algorithm (these include, amongst others, the Beethoven Symphony No. 5, the Mozart Oboe Quartet KV370, the First Mephisto Waltz by Liszt and Schoenberg Op. 23 No. 3). To the latter, 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 2 for an overview of the complete dataset.

[3] Bösendorfer SE 290

Table 2. Pieces in Database

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	16	68,358	–	–
Total	183	435,670		

5.2 Results

We simulated the task of quickly recognizing a played piece and deciding on the exact position in the score by playing the audio performances in our database to the system. To simplify the experiments we first ran the note recognizer on the entire set of recordings and then fed the output systematically to the symbolic audio matcher – we will discuss the additional delay which would happen during the preprocessing step in our on-line system below. For the evaluation we initialized queries starting with only 1 note and incrementally added further notes detected by the note recognizer one by one until the information was sufficient for the system to return the 'correct' position.

For the evaluation a score position X is considered correct if it marks the beginning (+/- 1 second) of a score section that is identical in note content, over a time span the length of the query (but at least 30 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. We systematically did the experiments in steps of 1 second, up to 30 seconds before the end of the recording which amounts to 34,841 recognition experiments in total.

Table 3 shows the results of this experiment, giving both statistics on the performance time in seconds and the 'time in number of recognized notes' it took the system until it first reported the correct position in the score. Of course this still involves a large degree of uncertainty as the system may again decide on another, incorrect, position when provided with the next recognized note. Thus we took the same measurements again with the constraint that the correct position has to be reported by the system 5 times in row, which shows that the system is confident and really settled on this position (see Table 4).

In general the algorithm returns the correct score position very quickly (e.g., in 50% percent of the cases it has to listen to the performance for only 1.87 seconds or less to confidently find the correct position). The algorithm never failed to come up with the correct position, and only in a few rare cases was it reported back with a big delay (e.g., the worst delay in Table 4 amounts to 45.28 seconds, but actually in 99% of the cases the delay was smaller than 11.5 seconds).

In a live setting (i.e., when the system is listening to an actual ongoing live performance) the additional constant lag due to the note recognizer would amount to about 210 ms (caused by needed window sizes for this transcription step). Additionally each query takes a certain amount of time which depends on the query size (see Table 5). So for a query of size 30 the total delay of the system on the described database amounts to about 235 ms.

In our opinion these are fantastic results which even experts in classical music would struggle to achieve (unfortunately we are not aware of any study on this matter). We will demonstrate this live at the conference.

Table 3. Evaluation results in detail (see text). This table gives the duration of the performance both in time and in detected notes until the system first reported the correct position in the database.

	Time	Notes
Best	0.16 sec	4
1^{st} Decile	0.53 sec	6
2^{nd} Decile	0.70 sec	7
3^{rd} Decile	0.87 sec	8
4^{th} Decile	1.06 sec	9
Median	1.27 sec	9
6^{th} Decile	1.53 sec	10
7^{th} Decile	1.88 sec	12
8^{th} Decile	2.47 sec	15
9^{th} Decile	3.76 sec	22
Worst	41.68 sec	417

Table 4. Evaluation results in detail (see text). This table gives the duration of the performance both in time and in detected notes until the system reported the correct position in the database five times in a row.

	Time	Notes
Best	0.31 sec	8
1^{st} Decile	0.84 sec	10
2^{nd} Decile	1.07 sec	11
3^{rd} Decile	1.30 sec	12
4^{th} Decile	1.57 sec	13
Median	1.87 sec	13
6^{th} Decile	2.22 sec	14
7^{th} Decile	2.67 sec	16
8^{th} Decile	3.35 sec	19
9^{th} Decile	4.78 sec	26
Worst	45.28 sec	421

6 CONCLUSION

In this paper we presented another step towards our goal, 'the ultimate classical music companion'. We proposed a system based on

Table 5. Mean query times for different query sizes

Query Size	Time
5 notes	3.02 ms
10 notes	10.83 ms
20 notes	19.37 ms
30 notes	24.29 ms
40 notes	28.05 ms
50 notes	33.74 ms
60 notes	38.66 ms
70 notes	43.79 ms

a combination of music transcription and symbolic fingerprinting which is able to detect almost instantly which piece a performer is playing, and the according position in the score.

The next step now is to include the proposed algorithm into our on-line tracker and make the complete system usable for musicians. In the near future we will further augment the repertoire of our system. Currently we are preparing the complete Beethoven piano sonatas (the "New Testament" of the piano literature) for our database. Regarding the scalability of our solution we foresee no problems, especially as the algorithm which inspired our symbolic fingerprinting solution [8] is used commercially with databases consisting of millions of songs[4].

7 ACKNOWLEDGEMENTS

This research is supported by the Austrian Federal Ministry for Transport, Innovation and Technology, and the Austrian Science Fund (FWF) under project number TRP 109-N23 and by the Austrian Science Fund (FWF) under project numbers Z159 and P22856-N23.

REFERENCES

[1] A. Arzt and G. Widmer, 'Towards effective 'any-time' music tracking', in *Proceedings of the Starting AI Researchers' Symposium (STAIRS 2010)*, (2010).

[2] A. Arzt, G. Widmer, and S. Dixon, 'Automatic page turning for musicians via real-time machine listening', in *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, (2008).

[3] S. Böck and M. Schedl, 'Polyphonic piano note transcription with recurrent neural networks', in *Proceedings of the 37th International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2012)*, (2012).

[4] V. Emiya, R. Badeau, and B. David, 'Multipitch estimation of piano sounds using a new probabilistic spectral smoothness principle', *IEEE Transactions on Audio, Speech, and Language Processing*, **18**, 1643–1654, (August 2010).

[5] 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).

[6] J. Haitsma and T. Kalker, 'A highly robust audio fingerprinting system', in *Proceedings of the Third International Symposium on Music Information Retrieval (ISMIR 2002)*, volume 2002, (2002).

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

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

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

[4] The algorithm is used by the Shazam service (http://www.shazam.com/)