THE MATCH FILE FORMAT: ENCODING ALIGNMENTS BETWEEN SCORES AND PERFORMANCES

Francesco Foscarin*
Johannes Kepler University
francesco.foscarin@jku.at

Emmanouil Karystinaios*
Johannes Kepler University
emmanouil.karystinaios@jku.at

Silvan David Peter*
Johannes Kepler University
silvan.peter@jku.at

Carlos Cancino-Chacón*
Johannes Kepler University
carlos_eduardo.
cancino_chacon@jku.at

Maarten Grachten
Independent Researcher
maarten.grachten@gmail.com

Gerhard Widmer
Johannes Kepler University
gerhard.widmer@jku.at

Abstract

This paper presents the specifications of *match*: a file format that extends a MIDI human performance with note-, beat-, and downbeat-level alignments to a corresponding musical score. This enables advanced analyses of the performance that are relevant for various tasks, such as expressive performance modeling, score following, music transcription, and performer classification. The match file includes a set of score-related descriptors that makes it usable also as a bare-bones score representation. For applications that require the use of structural score elements (*e.g.,* voices, parts, beams, slurs), the match file can be easily combined with the symbolic score. To support the practical application of our work, we release a corrected and upgraded version of the Vienna4x22 dataset of scores and performances aligned with match files.

*equal contribution.

Introduction

In this work, we present *match*: a file format for a complete and robust encoding of symbolic music alignments. The term *symbolic* refers to the class of musical data types that explicitly represent a set of elements from common music notation. Such a set must include at least note elements with their temporal position and (where applicable[1]) pitch and duration. Common data types that are symbolically encoded are musical scores (in MEI, MusicXML, Humdrum **kern, and MIDI[2] formats) and performances (in MIDI format). This is opposed to other data types, such as audio and raster score images, which only represent low-level information, such as amplitude over time, and pixel RGB values, respectively. Several works in music information retrieval (MIR) make use of symbolic data types as we can expect a more explicit representation of music to produce more efficient and musically interpretable systems.

A symbolically encoded performance, in short, *symbolic performance*, consists of a sequential representation of notes with a position and duration given in terms of real physical time. It is produced by recording a human performance with musical instruments fitted with proper sensors, such as MIDI keyboards, MIDI drums, Disklavier grand piano, and, to some extent, guitars with MIDI pickups. Those instruments can capture and explicitly represent the time and dynamic deviations that are natural aspects of an expressive performance (Honing, 2001). On the other side, a *symbolic musical score* expresses note positions in *musical units* such as fractions of quarter notes and beats and arranges them in temporal and organizational structures such as measures, beats, sub-beats, parts, and voices. It also explicitly represents dynamics and temporal directives and other high-level musical features such as time signature, pitch spelling, and key signatures.

The structured representation of the score and the expressive nuances encoded in the performance are complementary elements that can be combined to enable advanced musical analysis. Typical MIR tasks that benefit from it are expressive performance modeling, score following, music transcription, and performer classification. However, to fully leverage this information, we need an alignment between each corresponding element in a score and a performance (see Figure 1 for a short example). Some fully automated techniques have been proposed to produce alignments at note-level (Chen, Jang, & Liou, 2014; Gingras & McAdams, 2011; Nakamura, Yoshii, & Katayose, 2017), but, as they struggle in certain situations, it is still common to go through a manual annotation/correction phase performed by experts (Foscarin, McLeod, Rigaux, Jacquemard, & Sakai, 2020).

In this paper, we assume a score-performance alignment is given, for example as a result of a manual or semi-automatic alignment, and we address the problem of encoding it in a format that focuses on completeness, usability, and robustness. We name this format *match*. Completeness is ensured by explicit handling of repetitions structures and by including in the match file the entire list of notes in the performance, even if they are not present in the score, for example as a result of embellishments, player mistakes, or incomplete scores. For usability, we reduce the technical difficulties of operating across multiple files, by encoding in the match file a lossless representation of the performance, enhanced with a bare-bones representation of the score. This enables the usage of match files as a stand-alone representation for all tasks that focus on pitch and duration information. Furthermore, the extra

[1] Percussive notes may not have a pitch or duration.

[2] Though MIDI can only encode a partial set of the score information.

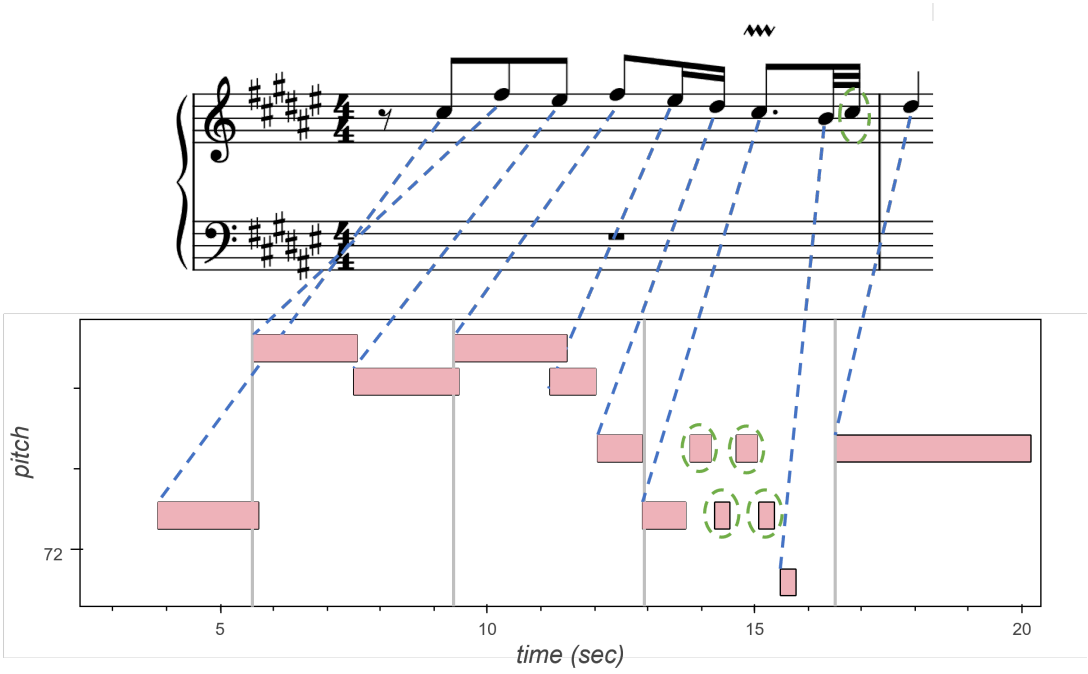

Figure 1: An example note-level alignment between a score and a performance of a score with a real piano performance. From Bach Fugue 13 in F sharp major, BWV 858. Aligned notes are connected blue dotted lines, while notes that are only in the score (deletion) or in the performance (insertion) are circled in green. Beat positions are marked with grey vertical lines in the performance.

score information acts as a redundancy safety layer to improve the robustness of the alignments for applications that need to link to the symbolic score to use other score elements (e.g., voices, parts, beams, slurs). On the contrary, a more naive encoding that only points at score positions with beats and measures number would fail in case of minor score modifications such as the splitting or time modification of a measure.

The Match format is text-based, sequentially structured, and human understandable. This enables the visual inspection and manual editing of its content, and eases its integration in different applications. For Python-based research, the usage of match files is further simplified by the Partitura package (Grachten, Cancino-Chacón, & Gadermaier, 2019) that offers off-the-shelf parsing and processing of this format. To support the practical application of our work, we release a corrected and upgraded version of the Vienna 4x22 dataset of symbolic performances and aligned scores (Goebl, 1999).

The remainder of this paper is organized as follows: in Section 1 we compare with other relevant research and highlight how our system can solve typical problems in this field. In Section 2 we detail the match file format and Section 3 we present the dataset and match file parsing with Partitura. Finally, in Section 3.2 we draw some conclusions and discuss future work.

1 Related work

Some datasets exist that contain alignments between scores and performances. For example, the expressive performance dataset (Marchini, Ramirez, Papiotis, & Maestre, 2014)

contains note-level alignments (onset, offset and pitch), the Mazurka dataset (Cook, 2007) provides beat-level alignments and the ASAP dataset (Foscarin et al., 2020) contains alignments at the measure level. To store them, these datasets use text files (one file for each monophonic part), Excel spreadsheets, and JSON files, respectively. None of those formats is directly extensible to include a complete set of alignments. The absence of a unique encoding forces researchers to spend time learning new formats and writing new code every time they need to target a different dataset.

A proposal for a general and easy-to-use encoding of note alignments is made by Devaney and Gauvin (2019), by extending the musical score (in MEI or Humdrum **kern format) with some performance-related information. A problem that arises from this approach is about encoding efficiency: if we are considering multiple performances of the same score, we would need to make multiple duplicates of the entire musical score. This creates a lot of repeated information, increases memory usage, and produces consistency problems if we want, for example, to correct a notation error in the score. Another problem is that this format can't encode information on performed notes that are not on the score, *e.g.,* embellishments or player mistakes. Similarly to this approach, our match file contains enough information on the score and performance to be used as a stand-alone representation for many tasks. However, instead of extending the score, we work in the opposite direction by extending the performance with a bare-bones score representation.

Nakamura et al. (2017) present a system that is able to automatically align two MIDI performances or a MusicXML score and a performance. The output of such a system is a list of references, similar to the main part of our match file. Differently from our approach, however, Nakamura uses real numbers with (6 decimal digits precision) to identify the onset and offset of notes in the MIDI file. This limited precision can lead to rounding problems and force the usage of approximate equality functions to retrieve the corresponding note in the MIDI. Instead, our approach ensures a lossless encoding of performance time positions with MIDI ticks that are already used in the MIDI file.

Older versions of the match file have been around for some time[3] but no official reference or documentation is available. The initial format was created to encode manually annotated note alignments between pieces performed on computer-monitored pianos such as Boesendorfer SE 290 and their corresponding scores. It was developed in *prolog*, a popular language at the time for database creation and retrieval of information. For this reason, each event or entry in the match file format is represented on a new line with a dot determining the end of a prolog-like compound term. In this paper, we propose an updated version of the match file (version 1.0.0) that includes support for repetitions and time point alignments (*e.g.,* beats and downbeats) and solves other small issues that were found in practical usage. This paper has among its goals to be a formal reference to this encoding format for a more widespread utilization in the research community.

2 Match files

Match is a text-based file format developed to encode a complete and robust alignment between a symbolic performance and a corresponding musical score. We start the description of the format with a high-level perspective on how it handles alignments and repetition

[3] `http://dx.doi.org/10.21939/4X22`

structures. Then we detail how the information is encoded.

2.1 Note and time alignment

Match files contain alignments at two levels: notes and time points (*e.g.,* beat and measure).

For note-level alignments, the match file encodes a mapping $match$ between the notes in a performance and the corresponding one in a score. Formally, let us consider the sets P and S of all notes in a symbolic performance and the corresponding musical score. $match$ is a partial function over P and S. For a performance that aligns perfectly with a score, $match$ becomes a total function, *i.e.,* it is defined $\forall e \in P$. However, due to player mistakes, embellishments, or incomplete scores, it may happen that some events in P do not have a corresponding event in S. The partial function is non-surjective, as there are multiple events in the score (*e.g.,* time signature annotations, barlines, etc.) that are not in the performance and there is the possibility of omitted notes by the performer. Moreover, $match$ can be non-injective if there are repetitions, where multiple events in the performance map to the same element in the score.

The case of time alignments is different because both score and performance time are continuous domains and a function between them cannot be easily annotated by music experts. Time level alignments can be seen as a set of samples from this function. Creators of the match files are free to select the granularity level they want to encode, usually at the downbeat or beat level.

2.2 Repetition structures

Handling repetitions for a performance and a corresponding score is a complex task due to the arbitrary way they can be interpreted. For instance, a performer may play the entire "unfolded" piece, while another may skip some repetitions. Other works, *e.g.,* Foscarin et al. (2020), create different score versions by manually removing the repetition marks that are not played in the performance. However, this approach produces multiple scores of the same piece, which complicates the comparison of the related performances.

By using match files, we do not modify musical scores; instead, we encode in the match file an unfolded (or reduced) score that matches the aligned performance. This is paired with an explicit representation of repetitions based on *sections*.[4] A score is segmented in multiple sections by repetition signs such as left repeat, right repeat, volta start, volta end, and navigation directions such as al Coda, dal Segno, da Capo, *etc*. This simplifies the task of comparing performances of the same piece with different repetition structures.

2.3 File encoding

A match file consists of a sequence of lines, each ending with a dot, and there are five different types of lines. Figure 2 highlights examples with different colors. Global information lines (in pink) encode elements that are constant throughout the entire piece, for example, composer, performer, title, version, *etc*. The score property lines (in yellow) con-

[4] Sections in match files have a similar role to MEI `<section>` (and `<ending>`) elements; however, they cannot be nested, and their usage is limited to repetition structures.

```
info(matchFileVersion, 1.0.0).
info(composer,Bach)
info(piece,Fugue 13 BWV858)
info(midiClockUnits,480).
info(midiClockRate,500000).
scoreprop(timeSignature,4/4,1,0.0).
scoreprop(keySignature,F# Maj,1,0.0).
snote(n0,[C,#],5,1:1,1/8,1/8,0.5,1.0,[])-note(0,73,1104,1647,43,0,0).
stime(1:2,0,1,beat)-ptime([1620]).
snote(n1,[F,#],5,1:2,0,1/8,1.0,1.5,[])-note(1,78,1620,2180,51,0,0).
snote(n2,[E,#],5,1:2,1/8,1/8,1.5,2.0,[])-note(2,77,2160,2727,56,0,0).
stime(1:3,0,2,beat)-ptime([2704]).
snote(n3,[F,#],5,1:3,0,1/8,2.0,2.5,[])-note(3,78,2704,3308,55,0,0).
snote(n4,[E,#],5,1:3,1/8,1/16,2.5,2.75,[])-note(4,77,5,3217,3464,56,0,0).
snote(n5,[D,#],5,1:3,3/16,1/16,2.75,3.0,[])-note(5,75,5,3472,3716,55,0,0).
stime(1:4,0,3,beat)-ptime(3716).
snote(n7,[C,#],5,1:4,0,3/16,3.0,3.75,[])-note(6,73,5,3716,3949,58,0,0).
insertion-note(7,75,3972,4084,58,0,0).
insertion-note(8,74,41024186,61,0,0).
insertion-note(9,75,4221,4335,54,0,0).
insertion-note(10,73,4341,4425,4425,63,0,0).
snote(n8,[B,n],4,1:4,3/16,1/32,3.75,3.875,[])-note(11,71,4456,4542,55,0,0).
snote(n9,[C,#],5,1:4,7/32,1/32,3.875,4.0,[])-deletion.
stime(2:1,0,4,downbeat)-ptime(4752)
snote(n17,[D,#],5,2:1,0,1/4,4.0,5.0,[])-note(13,75,4752,5808,55,0,0).
section(0.0,4.0,0.0,4.0,[])
sustain(17140,31,0,0).
sustain(17160,49,0,0).
```

Figure 2: An example of match file corresponding to the alignments of Figure 1. Examples of different types of lines are color-coded for an easier understanding.

tain score elements that can change, such as key signatures, time signatures, and performance directives. Alignment lines can have two formats, `snote(*)-note(*)` (in blue) and `stime(*)-ptime(*)` (in grey), for note and time point alignments, respectively. Non-aligned notes, *i.e.,* notes only in the performance or only in the score, are encoded with the keywords insertion and deletion (in green). Furthermore, specific alignment lines are designated for ornaments and trills, where multiple performed notes can refer to the same score notes or even a score marking. Repetition section lines (in red) relate the unfolded score times to the original score times, and sustain pedal lines (in orange) encode pedal information .

Table 1 contains value specifications for the note alignment line:

```
snote(Anchor,[NoteName,Mod],Octave,Measure:Beat, Offset,Dur,OnsetInBeats,OffsetInBeats,ScoreAttrList)-
note(ID,MIDIpitch,Onset,Offset,Velocity, MIDIchannel, MIDItrack).
```

The `Anchor` string corresponds to existing identifiers if there are any in the musical score (*e.g.,* MEI note ids) or generated ones. A suffix can be added in case of repetitions; for example, a score note ”n23” in a repeated section, will be referred to as ”n23-1” the first time and as ”n23-2” the second. The full specifications of the other line types are available at `https://cpjku.github.io/matchfile`.

3 Practical applications

To support the usage of match files in practical applications, we present a way of parsing and processing those files in Python and an updated and corrected match file dataset.

Value name	Value type	Description
Anchor	string	Note identifier (suffix after "-" for repetition)
NoteName	string	Pitch class name in [C, D, E, F, G, A, B]
Modifier	string	Pitch modifier in ["", n, b, #, bb, x]
Octave	integer	Octave number in scientific pitch notation
Measure	integer	Measure number (starting at 1, 0 for anacrusis)
Beat	integer	Integer beat number of note onset (starting at 1)
Offset	fraction (int/int)	Offset from beat position (fraction of whole note)
OnsetInBeats	float	Onset position in contiguous beats
DurationInBeats	float	Duration in beats
ScoreAttributesList	string	Note attributes ("grace", "appoggiatura", etc.)
ID	integer	Note identifier
MIDIpitch	integer	Pitch 0-127
Onset	integer	Time in MIDI ticks of the note on message
Offset	integer	Time in MIDI ticks of the note off message
Velocity	integer	Note on velocity 0-127
Channel	integer	MIDI channel 0-15
Track	integer	MIDI track

Table 1: Values specifications for the alignment line in a match file.

3.1 Handling Match files

Partitura (Cancino-Chacón et al., 2022; Grachten et al., 2019), an open-source Python package, offers off-the-shelf parsing and processing of match files. Partitura creates dedicated Python objects that give easy access to the information encoded in the match file, as well as lists of dictionaries relating performance and score note identifiers. If external scores are required, Partitura can load MEI, MusicXML, and Humdrum **kern scores and link notes and temporal positions by using the alignments in the match file. Partitura objects can be easily converted to note array or pianoroll representations.

3.2 Dataset

The Vienna 4x22 was originally compiled by Goebl (1999) and consists of 4 excerpts of solo piano pieces, each performed by 22 pianists. The pieces are 21 bars of Chopin Opus 10, 45 bars of Chopin Opus 38, 36 bars (the exposition) of Mozart KV331, and the full 32 bars of Schubert D783 (opus 33 No. 15). All performances were recorded on Bösendorfer 290 SE Grand Piano as MIDI-like data and subsequently each played note matched to its respective score note. We release an updated and corrected version of this dataset encoded in the current version of the match format at `https://github.com/CPJKU/vienna4x22`.

Conclusion and Future Work

In this paper, we proposed the match format for a complete and robust encoding of the alignments between symbolically encoded musical scores and performances. Note-level align-

ments and time-level alignments (*e.g.,* beat and measure) are supported. Match files can be used as a stand-alone representation of score and performance to reduce the technical difficulties of operating across multiple files. A Python package that can read and write match files is available.[5] We also release an updated and corrected version of the Vienna4x22 dataset that contains scores and performances aligned with match files.

The match file format is actively under development. This paper marks the release of the stable version 1.0.0 but further modifications are to be expected to support more features and solve problems that arise from its usage in practical applications. Other future works involve a graphical utility for the visualization and modification of alignments and tools to create them automatically (at least partially) given a corresponding score and performance.

Acknowledgements

This project receives funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement No 101019375 (*Whither Music?*).

References

Cancino-Chacón, C. E., Peter, S. D., Karystinaios, E., Foscarin, F., Grachten, M., & Widmer, G. (2022). Partitura: A Python Package for Symbolic Music Processing. In *Proceedings of the Music Encoding Conference (MEC2022).* Halifax, Canada.

Chen, C.-T., Jang, J.-S. R., & Liou, W. (2014). Improved score-performance alignment algorithms on polyphonic music. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (p. 1365-1369). Florence, Italy.

Cook, N. (2007). Performance analysis and Chopin's mazurkas. *Musicae Scientiae*, *11*(2), 183–207.

Devaney, J., & Gauvin, H. L. (2019). Encoding music performance data in Humdrum and MEI. *International Journal on Digital Libraries*, *20*(1), 81–91.

Foscarin, F., McLeod, A., Rigaux, P., Jacquemard, F., & Sakai, M. (2020). ASAP: a dataset of aligned scores and performances for piano transcription. In *International society for music information retrieval conference (ISMIR)* (pp. 534–541). Montréal, Canada.

Gingras, B., & McAdams, S. (2011). Improved Score-performance Matching Using Both Structural and Temporal Information from MIDI Recordings. *Journal of New Music Research*, *40*(1), 43–57.

Goebl, W. (1999). *The Vienna 4x22 Piano Corpus.* Retrieved from `http://dx.doi.org/10.21939/4X22` doi: 10.21939/4X22

Grachten, M., Cancino-Chacón, C. E., & Gadermaier, T. (2019). partitura: A python package for handling symbolic musical data. In *Late-Breaking Demo Session of the International Society for Music Information Retrieval Conference.* Delft, Netherlands.

Honing, H. (2001). From time to time: The representation of timing and tempo. *Computer Music Journal*, *25*(3), 50–61.

Marchini, M., Ramirez, R., Papiotis, P., & Maestre, E. (2014). The sense of ensemble: a machine learning approach to expressive performance modelling in string quartets.

[5] `https://partitura.readthedocs.io/`

Journal of New Music Research, *43*(3), 303-317.

Nakamura, E., Yoshii, K., & Katayose, H. (2017). Performance Error Detection and Post-Processing for Fast and Accurate Symbolic Music Alignment. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)* (pp. 347–353). Suzhou, China.