partitura: A PYTHON PACKAGE FOR HANDLING SYMBOLIC MUSICAL DATA

Maarten Grachten\textsuperscript{1} Carlos Cancino-Chacón\textsuperscript{2} Thassilo Gadermaier\textsuperscript{3}
\textsuperscript{1} Independent Researcher, Barcelona, Spain
\textsuperscript{2} Austrian Research Institute for Artificial Intelligence, Vienna, Austria
\textsuperscript{3} Institute of Computational Perception, Johannes Kepler University Linz, Austria
maarten.grachten@gmail, carlos.cancino@ofai.at, thassilo.gadermaier@jku.at

EXTENDED ABSTRACT

In this work we present partitura, a Python package for handling the symbolic musical information that is conveyed by modern staff notation. The package was born out of a need to process richly structured musical information in a less reductive way than the pianoroll representation that is very common in MIR, in which a score is represented as a list of timed pitch events. Although there are certainly valid use cases for pianoroll representations of music, we believe that some musical tasks can be more effectively addressed based on a richer data representation. Computational modeling of musical expression is one such task.

Musical scores contain a variety of musically relevant information that is typically not present in a pianoroll representation, including but not limited to pitch spelling, metrical structure, phrasing, voicing, articulation, tempo, dynamics, and musical form. A challenge when dealing with this information is that it requires more complex data structures than the matrix structure typically used to represent pianorolls. The partitura package uses the notion of a timeline to express the temporal scope of the elements in a score, such as notes, rests, slurs, measures, time and key signatures, and performance directions. Elements may contain references to each other. For example, a slur contains references to the starting and ending note of the slur. This approach is further illustrated below.

The package supports exporting and importing musical scores to and from files in MusicXML and MIDI format. Although the MIDI format in itself does not retain much of the musical information that partitura intends to capture, the package includes proven algorithms for pitch spelling, voice estimation, and key estimation (see below), to reconstruct some of that information.

In relation to the well-known music21\textsuperscript{1} Python package it should be noted that the aims of partitura are more modest. Whereas music21 provides a toolkit for computer-aided musicology—including functionality like visualization and searching corpora—partitura aims to facilitate processing musical information in Python. It roughly follows MusicXML in terms of musical entities, but as opposed to MusicXML, where time is largely implicit, partitura takes a strongly time-oriented approach. This approach allows for extracting local musical contexts in full detail, but makes it equally straightforward to extract subsets of information from the score as a whole.

In partitura a score is defined at the highest level by one or more Part objects, possibly grouped by PartGroup objects. Parts are typically associated with instruments, and each part may have one or more staves. Each Part contains a TimeLine object that encapsulates a sequence of TimePoint objects, each denoting a temporal position in the score (in an attribute $t$). A musical element such as a Note is added to the TimeLine by registering it with the TimePoints corresponding to its start and end positions. A particularly important element is the Divisions element, because it specifies the relation between the time interval $t_{p2} - t_{p1}$ between two timepoints $t_{p1}$ and $t_{p1}$, and the duration of a quarter note. Figure 1 shows a schematic representation of a Part object and its components.

\textsuperscript{1}https://web.mit.edu/music21/

Figure 1. Schematic representation of a Part object: A part contains a TimeLine object, which holds TimePoints (i.e., pegs that fix score elements in time). The blue lines represent the starting times of the objects in the score and the red lines represent the end times.

As mentioned above, partitura includes some tools for music analysis which are intended to fill in missing information with plausible values, for instance when loading a score from a MIDI file. For estimating the key signature of a piece, we use the Krumhansl–Shepard key identification algorithm [2]. We include an implementation of the ps13s1 algorithm [3] for estimating pitch spelling. For estimating voice information, we use VoSA [1], a contig mapping approach for voice separation in polyphonic music. To our knowledge, this is the first publicly available Python implementation of ps13s1 and VoSA.

The package is available on GitHub², with documentation available at readthedocs.orgʒ. Future work will include support of the MEI format⁴ and the match format⁵ which is used to encode performance-to-score alignments.

ACKNOWLEDGMENTS

This research has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 670035 (project “Con Espressione”) and the Austrian Science Fund (FWF) under grant P 29840-G26 (project “Computer-assisted Analysis of Herbert von Karajan’s Musical Conducting Style”).

REFERENCES


²https://www.github.com/mgrachten/partitura  
⁴https://music-encoding.org 
⁵http://www.eecs.qmul.ac.uk/~simond/match/