INTUITIVE ANALYSIS, CREATION AND MANIPULATION OF MIDI DATA WITH `pretty_midi`

Colin Raffel
LabROSA, Columbia University
Department of Electrical Engineering

Daniel P. W. Ellis
LabROSA, Columbia University
Department of Electrical Engineering

1. PARSING MIDI DATA

Despite being over 30 years old, the Musical Instrument Digital Interface (MIDI) standard remains in wide use [5]. This is likely due to its compactness, completeness, and widespread adoption. While MIDI is both a hardware and software standard, in the present work we will be focusing specifically on MIDI files. In a naive view, MIDI files can be seen as a bitwise representation of a musical score, with different bit sequences indicating musical events. As such, manipulating MIDI data directly is cumbersome, and it's no surprise that many software libraries have been developed for programmatically parsing, manipulating, and performing high-level analysis on MIDI data [1, 3, 6, 7].

We believe that the most intuitive representation is a hierarchical one, consisting of a list of instruments, each of which contains a sequence of events (notes, pitch bends, etc.). This is analogous to a per-instrument piano roll, the visualization commonly used for manipulating MIDI data in digital audio workstations. In addition, the timing of an event in MIDI is represented by tempo-dependent "ticks" relative to the previous event, making direct interpretation in terms of absolute time (in seconds) difficult.

All of the existing software libraries either represent MIDI data in a lower-level (directly corresponding to the bit-level representation) or a higher-level manner (only as musical features). This can make simple manipulations and analysis either require a great deal of source code and expertise or, in the case of modifications to aspects not supported by an abstract library, impossible. For example, in the python-midi module, [1] shifting up the pitch of all notes in a MIDI file by 2 semitones takes only a few lines of code. However, constructing a piano roll representation can take a few hundred lines of code because MIDI ticks must be converted to time in seconds, note-on events must be paired with note-offs, drum events must be ignored, and so on.

Based on these issues, we created the Python module `pretty_midi` for creating, manipulating and analyzing MIDI files. It is intended to make the most common operations applied to MIDI data as straightforward and simple as possible. `pretty_midi` represents a MIDI file in the hierarchical manner seen in Figure 1. The module includes functionality for parsing and writing MIDI files, creating and manipulating MIDI data, synthesis, and information extraction. Its source code is also intended to be straightforward to understand and modify and is available on GitHub. [2] In the following two sections, we outline the design and usage of `pretty_midi`. For up-to-date usage examples, please refer to `pretty_midi`'s documentation. [3]

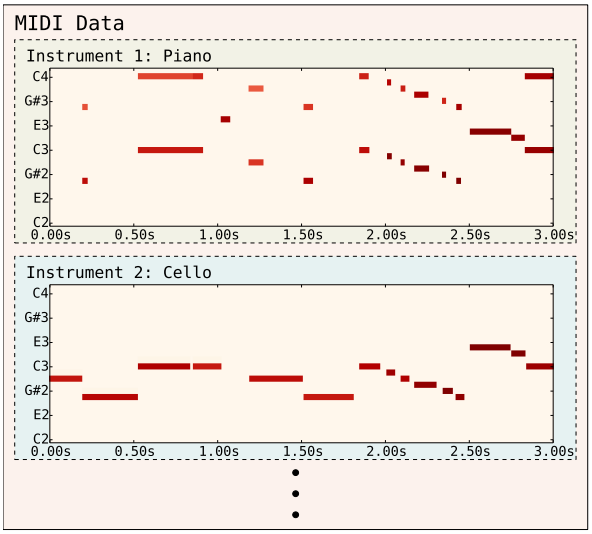

Figure 1. In `pretty_midi`, MIDI data is represented as a list of instruments, each of which has its own "piano roll" of events.

2. FUNCTIONALITY

As seen in Figure 1, `pretty_midi` represents MIDI data as a hierarchy of classes. At the top is the `PrettyMIDI` class, which contains global information such as tempo changes and the MIDI resolution. It also contains a list of `Instrument` class instances. Each `Instrument` is specified by a program number and a flag denoting whether it is a drum instrument. `Instrument` class instances also

*Please direct correspondence to craffel@gmail.com
[1] http://github.com/vishnubob/python-midi

[2] http://github.com/craffel/pretty-midi
[3] http://craffel.github.io/pretty-midi/

contain three lists, one each for `Note`, `PitchBend`, and `ControlChange` class instances. The `Note` class is a container for MIDI notes, with velocity, pitch, and start and end time attributes. Similarly, the `PitchBend` and `ControlChange` classes simply have attributes for the bend or control change's time and value.

2.1 File I/O

The top-level `PrettyMIDI` class can be instantiated with a path to an existing MIDI file, in which case the class will be populated by parsing the file. It can also be instantiated without a pre-existing file for creating MIDI data from scratch. For output, the `PrettyMIDI` class has a `write` function which exports its data to a valid MIDI file.

2.2 Information Extraction

`PrettyMIDI` class instances have functions for performing analysis on the data they contain, some of which have a corresponding function in the `Instrument` class. Some of the implemented functions include:

- `get_tempo_changes`: Returns a list of the times and tempo (in BPM) of all MIDI tempo change events
- `estimate_tempo`: Returns an empirical tempo estimate according to inner-onset intervals, as described in [2]
- `get_beats`: Returns a list of beat locations by using the MIDI tempo change events for tempo and estimating the first beat as described in [2]
- `get_onsets`: Returns a list of all of the onsets (start times) of each MIDI note
- `get_piano_roll`: Returns a piano roll matrix representation of MIDI notes, as visualized in Figure 1
- `get_chroma`: Computes chroma features (also known as pitch class profile) [4] of the MIDI data based on the piano roll representation

2.3 Synthesis

In `pretty_midi`, MIDI data can be synthesized as audio using either the `synthesize` or `fluidsynth` methods. `synthesize` uses a periodic function (e.g. sin) to synthesize the each note, while `fluidsynth` utilizes the Fluidsynth program [4] which performs General MIDI synthesis using a SoundFont file. Both methods return arrays of audio samples at some specified sampling rate.

2.4 Utility Functions

`pretty_midi` has utility functions for converting between representations of MIDI notes (name, note number, frequency in Hz, and drum name for percussion instruments), program number and instrument name/class (according to the General MIDI standard) and pitch bend value and semitones. Each `PrettyMIDI` class instance can also readily convert between MIDI ticks and absolute seconds. These functions allow for semantically meaningful creation and representation of MIDI data.

[4] `www.fluidsynth.org`

3. FUTURE WORK

While `pretty_midi` contains some functions for extracting information from MIDI data, it does not currently implement all of the feature extraction functionality included in jSymbolic [6]. It also currently is missing the ability to do some higher-level subjective musicological analysis tasks, such as extracting chords and estimating the beats in the absence of tempo change events. This type of analysis exists in the MATLAB MIDI Toolbox [3] and the Melisma Music Analyzer [7]. As this functionality is added in the future, `pretty_midi` will further satisfy the need for a general-purpose and easy-to-use software package for MIDI creation, analysis, and manipulation.

4. ACKNOWLEDGMENTS

The authors would like to thank Douglas Repetto, Dylan Kario, and Hilary Mogul for beta testing `pretty_midi`.

5. REFERENCES

[1] Michael Scott Cuthbert and Christopher Ariza. music21: A toolkit for computer-aided musicology and symbolic music data. In *Proceedings of the 11th International Conference on Music Information Retrieval*, pages 637–642, 2010.

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

[3] Tuomas Eerola and Petri Toiviainen. MIR in Matlab: The MIDI toolbox. In *Proceedings of the 5th International Conference on Music Information Retrieval*, pages 22–27, 2004.

[4] Takuya Fujishima. Realtime chord recognition of musical sound: a system using common lisp music. In *Proceedings of the International Computer Music Conference*, pages 464–467, 1999.

[5] Jim Heckroth. *The complete MIDI 1.0 detailed specification: incorporating all recommended practices*. MIDI Manufacturers Association, 1996.

[6] Cory McKay and Ichiro Fujinaga. jSymbolic: A feature extractor for MIDI files. In *Proceedings of the International Computer Music Conference*, pages 302–305, 2006.

[7] Daniel Sleator and David Temperley. The melisma music analyzer. `http://-www.link.cs.cmu.edu/music-analysis`, 2001.