Cosmodoit: A Python Package for Adaptive, Efficient Pipelining of Feature Extraction from Performed Music

Corentin Guichaoua [1], **Daniel Bedoya** [1], **and Elaine Chew** [1,2]

1 STMS UMR9912, CNRS / Ircam / Sorbonne Université / Ministère de la Culture, France **2** King's College London, United Kingdom

Summary

Computational analysis of performed music is an essential part of music information research (MIR). Almost all the music that we hear has been shaped through performance. Music performance analysis focuses on the representation and characterisation of acoustic variations performers introduce when communicating music (Lerch et al., 2020). It is sometimes referred to as musical prosody (Palmer & Hutchins, 2006). These acoustic variations are shaped by individual interpretations of music pieces. They encode performers' conceptualisations of music structures (Guichaoua et al., 2024) and provide valuable cues for listeners' response to music (Cotic et al., 2024).

Music performance analysis has grown considerably in recent years with the development of algorithms and tools in programming languages such as Matlab (Pampalk, 2004), C++ (Nakamura et al., 2017), and Python (Guo et al., 2019; Peter et al., 2023). Each algorithm or tool is tailored to perform a specific task out of several that are critical to the analysis of performed music. The diversity of implementation languages and data formats makes existing algorithms difficult to combine. There is an urgent need for a comprehensive and efficient tool that can streamline the process of feature extraction from performed music. This is the aim of Cosmodoit.

Cosmodoit is a novel Python package that combines performance-to-score alignment and symbolic and audio music feature extraction. The software is designed to be an adaptive and efficient tool that can process performed music at scale. The pipeline of Cosmodoit is designed to be flexible, allowing users to perform selective processing and compute only the desired features and the ones on which they depend, thus avoiding duplicated work and reducing computational time.

The design of Cosmodoit was based on several key considerations, including the requirement to simplify the process of music performance analysis, reducing errors caused by file changes or swaps, and providing a tool that can be updated regularly to accommodate evolving needs. The package is modular and extensible, allowing users to add or ignore features as needed. In addition, Cosmodoit allows for the computation of small increments (like adding features) as required, making it an ideal tool for both research and development purposes.

One of the key features of Cosmodoit is its ability to handle algorithms from a variety of programming languages, making it a versatile tool for music performance analysis. The package also has the ability to change parameters such as window length, allowing users to fine-tune the pipeline and extract the desired features from performed music while ensuring the consistency for all dependent features. This makes Cosmodoit a handy tool for researchers and developers in the field of music performance analysis.

Statement of need

Data collection in music performance can be a very slow process, involving inviting performers, recording sessions, and gathering additional data such as scores. As a result, new research on methods and insights is often intertwined with building datasets, which leads to a growing need for re-computing features because of new or updated data and changes to the downstream code and its parameters.

The process of combining features in music performance analysis has traditionally been a manual one, requiring researchers and developers to keep track of the order in which features are extracted and combined. This manual process can be time-consuming and error-prone, as updates to the data or the code can require that the entire pipeline be re-run or risk missing a dependency resulting in out-of-sync features. This is compounded by the variety of environments, as some steps may require running processes in proprietary software, while others are long calls from the command line. As software in the analysis pipeline are usually developed independently, the tools being combined in sequence often do not share common input and output formats, which introduces yet an extra step where errors can occur.

Design Choices

Python is a widely-used programming language in music performance analysis, and many existing algorithms and tools are available in this language. Building the pipeline in Python makes it easier for researchers and developers to integrate their existing code into the Cosmodoit framework, while enabling code wrapping from most other languages as well.

Cosmodoit was built using Doit (Schettino, 2018), a Python-based build system, as its foundation. Doit has a proven track record of being a reliable and efficient build system, having been used successfully in a variety of projects not limited to compiling code. On top of the dependency and up-to-date tracking of Doit, Cosmodoit dynamically handles the discovery of newly added performances and which tasks can be run based on available file types—e.g., skipping the score alignment when no score is available—such that updating the features can usually be done in a single-word command. The modular structure of the package also allows for selective processing, where the user can choose to process only specific features, reducing the computational time. This not only saves time but also allows for efficient exploration of evolving methods and datasets.

Cosmodoit currently contains three main submodules based on prior work: a new Python port of the loudness computation from Elias Pampalk's Matlab Music Analysis Toolbox (Pampalk, 2004), a wrapper module for Eita Nakamura's C++ MIDI-to-MIDI music alignment (Nakamura et al., 2017), and Rui Guo's Python implementation of Herremans & Chew (2016)'s Java-based harmonic tension computations in midi-miner (Guo et al., 2019). It also includes a basic beat finding algorithm based on the result of the alignment and small modules to extract note velocity and sustain pedal values from a performed MIDI file. Figure 1 shows a typical set of processes triggered by a single command-line call.
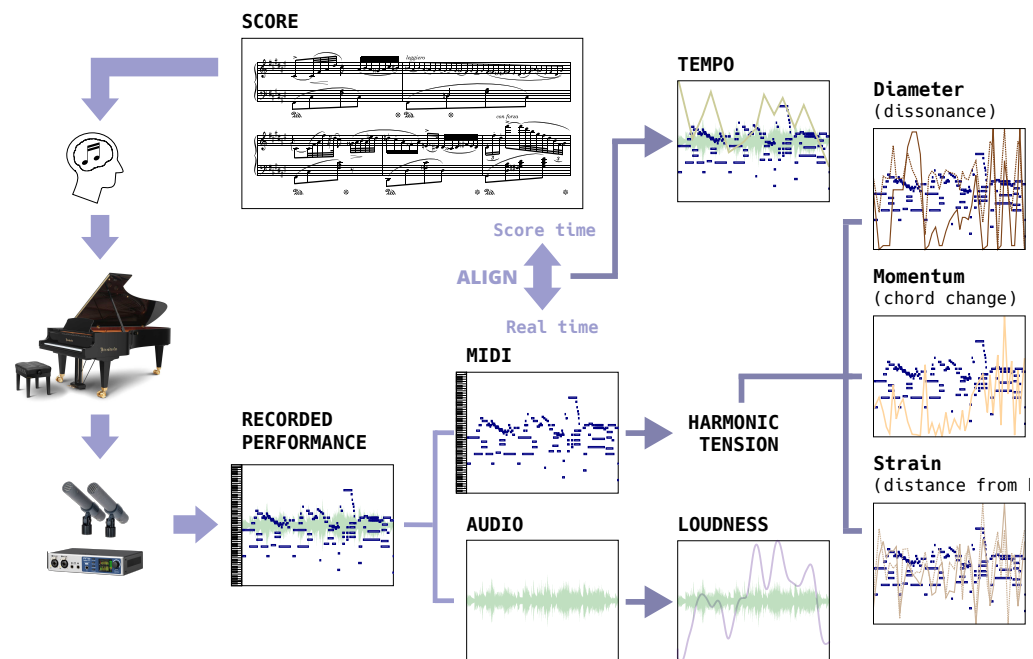
Figure 1: Cosmodoit system processes initiated by a single command-line call. Example based on Raoul Pugno's performance of Chopin's Nocturne Op.15 No.2 in F♯ major (Legendary Artists Collection courtesy of Bösendorfer). Cosmodoit's output data streams are visualised on the web based citizen science platform CosmoNote.

Where relevant, such as for the loudness computation, modules can expose parameters of the algorithm through a configuration file to override their default values at the dataset level, without risk of naming clashes. A template of such a file is provided in the repository, with all exposed parameters for the existing modules. Changing parameters will trigger the re-computation of affected features on the next run. While the current version does not contain alternative algorithms for any given feature, the parameter system can be leveraged to switch between different methods.

Finally, thanks to the modular structure of the package, new modules to compute additional features can easily be added with minimal modification of the dodo.py file, which serves as the main entry point. Provided the new module follows the template of existing modules, the corresponding tasks will be incorporated into the pipeline and have their dependencies detected and run automatically.

Acknowledgements

References

Cotic, N., Pope, V., Soliński, M., Lambiase, P., & Chew, E. (2024). A computational method for empirically validating synchronisation between musical phrase arcs and autonomic variables. *Proceedings of Computing in Cardiology (CinC2024)*.

Guichaoua, C., Lascabettes, P., & Chew, E. (2024). End-to-end bayesian segmentation and similarity assessment of performed music tempo and dynamics without score information. *Music & Science*, 7. https://doi.org/10.1177/2059204324123341

107 Guo, R., Herremans, D., & Magnusson, T. (2019). Midi miner – a python library for tonal
108   tension and track classification. *arXiv*, *1910.02049*. https://github.com/ruiguo-bio/
109   midi-miner

110 Herremans, D., & Chew, E. (2016). Tension ribbons: Quantifying and visualising tonal tension.
111   *Proceedings of the Second International Conference on Technologies for Music Notation*
112   *and Representation (TENOR)*, *2*, 8–18. https://dorienherremans.com/tension

113 Lerch, A., Arthur, C., Pari, A., & Gururani, S. (2020). An interdisciplinary review of music
114   performance analysis. *Transactions of the International Society for Music Information*
115   *Retrieval*, *3*(1), 221–245. https://doi.org/10.5334/tismir.53

116 Nakamura, E., Yoshii, K., & Katayose, H. (2017). Performance error detection and post-
117   processing for fast and accurate symbolic music alignment. *Proc. ISMIR*, 347–353.
118   https://midialignment.github.io/demo.html

119 Palmer, C., & Hutchins, S. (2006). What is musical prosody? *Psychology of Learning and*
120   *Motivation*, *46*, 245–278. https://doi.org/10.1016/S0079-7421(06)46007-2

121 Pampalk, E. (2004). A matlab toolbox to compute similarity from audio. *Proceedings of*
122   *the ISMIR International Conference on Music Information Retrieval (ISMIR'04)*. https:
123   //www.pampalk.at/ma

124 Peter, S. D., Cancino-Chacón, C. E., Foscarin, F., McLeod, A. P., Henkel, F., Karystinaios, E.,
125   & Widmer, G. (2023). Automatic note-level score-to-performance alignments in the ASAP
126   dataset. *Transactions of the International Society for Music Information Retrieval*, *6*(1),
127   27–42. https://doi.org/10.5334/tismir.149

128 Schettino, E. (2018). *Doit automation tool, 2008–*. http://pydoit.org