PREDICTING MUSIC HIERARCHIES WITH A GRAPH-BASED NEURAL DECODER

Francesco Foscarin[1] **Daniel Harasim**[2] **Gerhard Widmer**[1]
[1] Institute of Computational Perception, Johannes Kepler University Linz, Austria
[2] Independent Researcher
francesco.foscarin@jku.at

ABSTRACT

This paper describes a data-driven framework to parse musical sequences into dependency trees, which are hierarchical structures used in music cognition research and music analysis. The parsing involves two steps. First, the input sequence is passed through a transformer encoder to enrich it with contextual information. Then, a classifier filters the graph of all possible dependency arcs to produce the dependency tree. One major benefit of this system is that it can be easily integrated into modern deep-learning pipelines. Moreover, since it does not rely on any particular symbolic grammar, it can consider multiple musical features simultaneously, make use of sequential context information, and produce partial results for noisy inputs. We test our approach on two datasets of musical trees – time-span trees of monophonic note sequences and harmonic trees of jazz chord sequences – and show that our approach outperforms previous methods. [1]

1. INTRODUCTION

Tree-like representations are a powerful tool in many approaches to music analysis, such as Schenkerian Theory and the Generative Theory of Tonal Music (GTTM). In the Music Information Retrieval (MIR) literature, we find tree models of melodies [1–4], chord progressions [5–8], and rhythm [9–13]. Parallels between aspects of music and language are often drawn, as these have similar hierarchical properties and their underlying cognitive mechanisms could be closely related [14]. However, with a few exceptions, such as instrument grouping and metrical information in scores, music is generally encoded sequentially without explicit information about its hierarchical organisation. The task of creating such hierarchies from a sequential representation is called *parsing* and it is an active object of study in the MIR community [3, 7, 11, 15].

Current parsing approaches are based on *generative grammars*, typically context-free-grammars (CFG) or similar related mechanisms, which can be fundamentally seen as a set of expansion rules generating a tree from the top (the root) to the elements that compose the sequence (the leaves). Grammar rules can be enriched with a probability model that permits the ranking of different parses by plausibility. When a grammar is available, parsing can be achieved with grammar-based parsing algorithms, typically variants of the Cocke–Younger–Kasami (CYK) algorithm [16]. While the grammar rules are most often built by hand, by relying on musicologists' knowledge, the probabilities can be learnt from data if sufficient amounts of musical sequences with ground-truth tree annotations are available. The grammar approach has the strong advantage of leveraging an interpretable and cognitively plausible mechanism. Still, it has the following limitations: it is hard to achieve robustness against noisy data, which can cause a complete failure with no output in case the sequence cannot be produced by the grammar rules; it requires a high degree of domain knowledge; it is challenging to account for multiple musical dimensions in a single grammar rule; and parsing is usually so slow for long sequences that heavy pruning is necessary (CYK-parsing complexity is cubic in the length of the sequence, parallelisation does not help much, and there is no active research in developing dedicated hardware).

Inspired by recent research in the field of natural language processing (NLP), we propose a novel, *grammar-less* approach that requires little domain knowledge (only for the feature extraction phase), can easily consider multiple musical features and sequential information, produces partial results for noisy input, and is potentially scalable to longer sequences and larger datasets (since its components are proven to succeed in such scenarios). Our system works by predicting *dependency trees* which consist of dependency arcs between the input sequence elements. Such a structure can be used as-is or later be converted into *constituent trees* which are typically used to model music hierarchies (see Figure 1). The probability of each dependency arc is predicted in parallel (i.e., without considering other dependencies during prediction) by leveraging the rich contextual information produced by a transformer encoding of the input sequence. This set of probabilities is then run through a post-processing algorithm to ensure a valid tree structure (i.e., no cycles of dependency arcs).

We pair our Music Dependency Parser MuDeP with a

[1] All our code and data are publicly available at https://github.com/fosfrancesco/musicparser

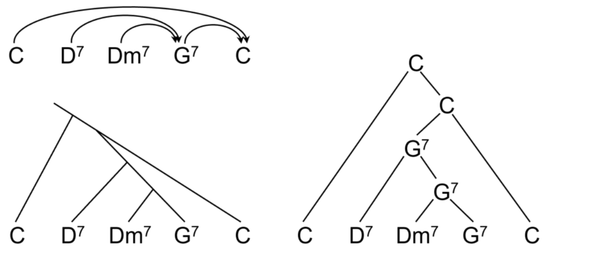


Figure 1. The tree harmonic analysis of the A Section of "Take the A Train" in three different representations. Top: dependency tree, Left: GTTM-style constituent tree. Right: CFG-style constituent tree.

procedure that enables its usage from constituent trees, and test it on two tree datasets: the time-span treebank from the GTTM database [17], which expresses subordinate relations between notes in monophonic melodies; and the Jazz Harmony Treebank (JHT), a set of harmonic analyses for chord sequences [18]. We compare the results of our system with the best-performing available approaches and obtain new state-of-the-art results.

2. RELATED WORK

Music Trees and Music Parsing. Trees of musical sequences have traditionally been notated as constituent trees [1–3, 5–13, 19], with few exceptions, such as the usage of a dependency-based evaluation metric [20], and the computation of pairwise voice dependencies [4, 21].

A system for parsing jazz chord sequences into harmonic analyses has been proposed by Harasim et al. [7] and later evaluated on a larger dataset [20]. We compare our results to this approach below. Automatic grammar-based parsing of time-span GTTM trees has been attempted by Hamanaka et al. [22, 23] and Nakamura et al. [2]. The latter obtained comparable results with an approach that doesn't require manual parameter tuning, and we compare our system with it. More recently, deep-learning-based approaches were also proposed [3, 24, 25] but the first two focus only on GTTM metrical and grouping information, and the latter focus mainly on evaluating the usage of time-span trees for melodic morphing and we could not reproduce their results.

Natural Language Parsing. Our model architecture is inspired by the graph-based dependency parser of Dozat and Manning [26, 27]. This model, extended with second-order dependency predictions [28] and pretrained language models [29], is still the state of the art for NLP sentence parsing [30]. Still, we make some substantial changes: the embedding layer is adapted to work from musical input, the encoder is a transformer instead of an LSTM, and, instead of the bilinear layer for arc prediction, we use a linear layer. All these choices are motivated by ablation studies.

3. TREE FORMATS FOR MUSIC ANALYSES

In this section, we detail the types of tree used in this paper, highlight their differences, and propose algorithms to translate between them.

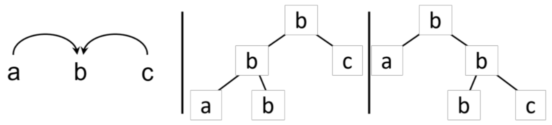


Figure 2. A dependency tree with double-sided dependencies (left). It corresponds to two possible constituent trees (middle and right).

3.1 Constituent vs Dependency Trees

A tree can be defined recursively as a node with an arbitrary number (including 0) of children that are also trees. The node that is not a child of another node in the tree is called *root*, the nodes that do not have children are called *leaves*, and the remaining nodes are called *internal nodes*. When a tree is used to model some relations of the elements of a sequence there are two possible configurations: *dependency trees*, where each node (leaf, internal, and root) represents one and only one element of the sequence; and *constituent trees* where all elements of the sequence are represented in the leaves, and root and internal nodes represent nested groupings of such elements.

Among the constituent trees there exist different representations. The bottom part of Figure 1 shows the two kinds we consider in this paper: the one introduced by Lerdahl and Jackendoff [31] in their Generative Theory of Tonal Music (GTTM), and the one built from the Context-Free-Grammar (CFG) of jazz harmony by Harasim et al. [7]. The two representations convey almost the same information: they are both binary trees (i.e., every node has either 0 or 2 children), the internal nodes are denoted by line intersections on the first, and by explicit labels on the second; they both specify an order of importance among the children (i.e., the choice of a *primary* and *secondary* child) by the straight line continuation, or by labelling the node with the label of the primary child. However, this latter mechanism cannot differentiate between primary and secondary when both children have the same label; therefore, the GTTM representation is slightly more informative.

Our approach does not directly treat constituent trees but considers dependency trees. Each child in such a tree is called *dependent*, and the node of which it is a child is called the *head*. Dependency trees can represent the same information as the binary constituent trees described above. Indeed, a dependent-head arc is equivalent to a head-labelled constituent node with two children: the primary is again the head, and the secondary is the dependent. There is only one ambiguity: the dependency tree does not encode a splitting order in the case of *double-sided dependencies*, a configuration in which one head has dependents on both sides. This makes the dependency-to-constituent transformation not unique (see Figure 2). This configuration is never present in our datasets (i.e., the root is always the left-most or right-most element in the sequence) thus we don't handle it. For more general datasets, one could add a binary classifier that predicts the splitting order.

The dependency trees built from the constituent trees are *projective*, i.e., for all their arcs $x_{\text{dep}} \rightarrow x_{\text{head}}$, there is a path from the head to every element x that lies between the head and the dependent in the sentence [32]. This means that there are no "crossing arcs", e.g., $x_1 \rightarrow x_3, x_2 \rightarrow x_4$.

Before proceeding with the paper, we introduce some notation we will use in the next sections. We denote the sequence that constitutes the input of our system as $x = [x_1, \ldots, x_\lambda]$, where λ is the sequence's length. We represent the dependency tree over x as the set of dependent-head [2] indices that corresponds to each arc $x_{\text{dep}} \rightarrow x_{\text{head}}$:

$$y = \{(\text{dep}, \text{head}) \mid \text{dep}, \text{head} \in [1, \ldots, \lambda]\} \tag{1}$$

3.2 Tree Conversion Algorithms

Since the ground-truth annotations in our datasets are constituent trees, we translate them into dependency trees for training. We also translate tree predictions back to constituent trees to run constituent-based evaluation metrics, and when we are interested in using such a representation as input for further applications. We assume our constituent trees to be binary trees and not contain double-sided dependencies. For simplicity, we consider CFG-style constituent trees with labels in their internal nodes.

3.2.1 Dependency to Constituent Tree

Existing NLP implementations of this transformation are unnecessarily complicated for our scenario because they consider compound node labels and double-sided dependencies [33]. Instead, we present a recursive top-down algorithm which yields a unique constituent solution for every single-sided dependency tree.

The algorithm takes a fully formed dependency tree and starts with the root of the (to-be-built) constituent tree. At each step, it removes one dependency and adds two new constituent nodes. The recursive function takes as input a dependency tree node and a constituent tree node, both labelled with the same sequence element. The constituent node gets assigned two children: the primary is labelled with the element of the input nodes, and the secondary is labelled with the dependent that is further away in the sequence. The choice of which is the left and the right child respects their label position in the sequence. The considered dependency is removed from the tree and the recursive function is called two times, once for each constituent child (with the corresponding dependency node). The process stops when the dependency tree node has no dependents.

3.2.2 Constituent to Dependency Tree

This algorithm was used in the literature (e.g., [18]). It starts from a fully formed constituent tree and a dependency tree without any dependency arcs, consisting only of the nodes labelled with sequence elements. The algorithm groups all internal tree nodes with their primary child (which all have the same label) and uses all secondary child relations originating from each group to create dependency arcs between the group label and the secondary child label.

[2] We indicate dependency arcs as arrows pointing in the direction of the head. Note that in other (NLP) papers, the opposite convention is used.

4. PARSING TECHNIQUE

Our goal is to predict a dependency tree y for a given musical sequence x. Our pipeline consists of three steps: feature extraction from x; prediction of dependency relations; and postprocessing to ensure that the output is a valid tree structure. In the training phase, the output (before postprocessing) is compared with the ground truth dependency tree and a loss is computed to update the model parameters via backpropagation.

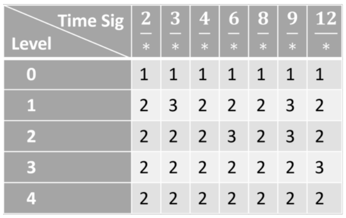
4.1 Feature Extraction

For each input element, $x_i \in x$, we produce three groups of features. The first is a "static" description of the element (i.e., without any temporal information), the second encodes the element's duration, and the third encodes the element's metrical position, i.e., its position in the measure relative to the hierarchy induced by the time signature. The static description is built differently for chords and notes, while the other two are independent of the input type. Note that, due to our model architecture (see next section), we need categorical features and it is not primarily important to keep their number small or to have them ordered.

For note sequences, the *static description* of each element is a single integer corresponding to either the MIDI pitch of the element in $[0, \ldots, 127]$ if it is a note or with the value 128 if the element is a rest. For chord sequences, we use three integers. The first in $[0, \ldots, 11]$ encodes the pitch-class of the chord root. The second in $[0, \ldots, 5]$ specifies the basic form of the chord among major, minor, augmented, half-diminished, diminished, and suspended (sus). The last in $[0, 1, 2]$ denotes a chord extension among 6, minor 7, or major 7. The chord labels were simplified by the author of the dataset to only include these extensions, but in a more general scenario, a larger set of integers could be used. The chord sequences do not contain rests.

We represent the *durations of the elements* with discrete values normalised by the duration of the measure. We precollect the list of all durations occurring in the dataset and encode each element's duration as an index on that list. For the GTTM dataset, this would be an integer in $[0, \ldots, 44]$, while for the JHB dataset, it is an integer in $[0, \ldots, 5]$. The number of possibilities is very different, since the temporal position of chords follows much simpler rules, mostly occurring only at the beginning or in the middle of a bar for simple time signatures and at three bar positions for compound time signatures. For tied notes, we consider a single note with the total duration, and notes can last more than one measure. This is different from the annotations in the JHT in which each measure opens a new chord symbol, even if the same chord is repeated in consecutive measures.

To represent the *metrical position*, we use an inverse measure of metrical strength, encoded with a single integer in $[0, \ldots, 5]$. This integer is computed as a function of the normalised temporal position in the measure $t \in [0, 1[$, and the time signature numerator. Each time-signature numerator is associated with a template of metrical divisions m, as proposed by Foscarin [10] and here extended to more time signatures. For example, a time signature with a nu-



Level \ Time Sig	2 *	3 *	4 *	6 *	8 *	9 *	12 *
0	1	1	1	1	1	1	1
1	2	3	2	2	2	3	2
2	2	2	2	3	2	3	2
3	2	2	2	2	2	2	3
4	2	2	2	2	2	2	2

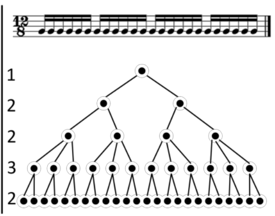


Figure 3. Left: metrical divisions m for different time signature numerators. Right: visualisation of metrical divisions for a measure with time signature 12/8.

merator 12 (e.g., 12/8 or 12/4) will have metrical divisions $m = [1, 2, 2, 3, 2]$, i.e., the whole measure at level 0 is divided into two parts at level 1, each resulting part is divided in 2 at level 2, then 3 at level 3, and 2 at level 4. Table 3 reports metrical divisions for all numerators we consider.

Each level l in the metrical division defines a temporal grid with step $\delta_l = 1/\prod_{l=0}^{l} m_l$, and the *inverse metrical strength* is defined as the lowest level for which the note position falls on the temporal grid, $\min_l(l \mid t/\delta_l \in \mathbb{N})$. For example, a time signature 6/8 defines grids with steps $[1, \frac{1}{2}, \frac{1}{6}, \frac{1}{12}, \frac{1}{24}]$, and the notes of the measure |♩♪♩.| will have normalised temporal position $[0, \frac{2}{6}, \frac{1}{2}]$ and inverse metrical strength $[0, 2, 1]$. If the note doesn't align with any temporal grid, then its inverse metrical strength is the maximum, 5 in our settings. Using metrical strength as input to our system may seem overly complicated. However, given the small size of our datasets and the high variety of time signatures, we need a mechanism to encode metrical information generalisable across different time signatures. It is to be expected that with a larger dataset size, this feature could be discarded, as the model could learn similar information from the list of notes with duration.

The feature extraction process lets us build the input matrix $\mathcal{X} \in \mathbb{N}^{\lambda \times \phi}$ where λ is the sequence length, and ϕ is the number of features for each element: 3 for the GTTM dataset, and 5 for the JHT dataset. Before moving on, it has to be noted that there exist other more general ways of transforming symbolic music into convenient inputs for deep learning models, notably the tokenisation techniques, e.g., [34, 35] inspired by NLP research. However, given the small dimension of our dataset and the fact that our melodies are strictly monophonic, we prefer to use a more compact, ad-hoc input representation. Our parsing framework remains general and usable with other techniques.

4.2 Model

Our model consists of two parts: an encoder and an arc predictor (see Figure 4). The goal of the encoder is to enrich the input features $\mathcal{X}$ with contextual information. The arc predictor uses the enriched sequence features to predict whether each possible pair of elements in the input sequence should be connected by a dependency arc.

The first part of the encoder is an embedding layer, a learnable look-up table which maps our collection of categorical features (each integer) to points in a continuous multidimensional space. Specifically, we use ϕ embedding

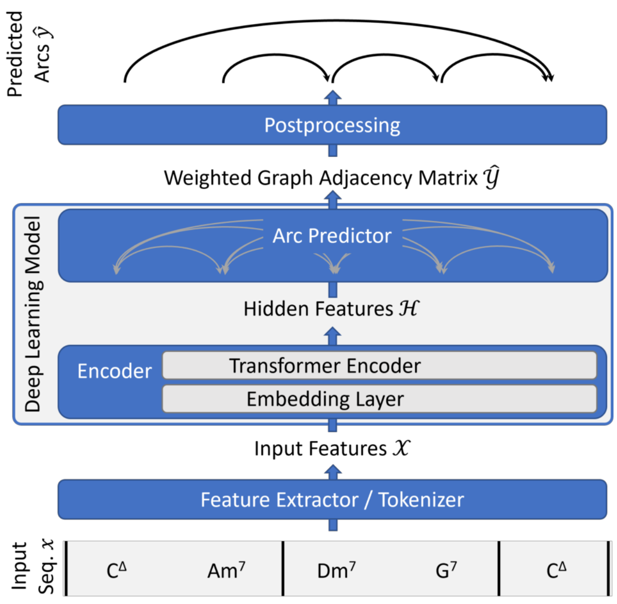


Figure 4. Architecture of our model. The input displayed is an example of a chord sequence, but the same architecture is used for note sequences.

layers (one for each input feature), which work independently, and map all values that the feature can have to a vector of a fixed embedding dimension. All vectors are then summed together to obtain a unique representation while keeping the input size small (see [36] for an explanation of why summing is better than concatenating). After the embedding layer, we have the encoder part of a transformer [37] with relative position representations [38]. It outputs a matrix with the same number of rows as the input matrix $\mathcal{X}$ (one for each sequence element) but with a (possibly) different number of hidden-feature columns h. Onto this, we concatenate a new learnable single row that acts as the head of the root node. The result is a new matrix $\mathcal{H} \in \mathbb{Q}^{(\lambda+1)\times h}$.

The arc predictor part of our model is a multilayer perceptron (MLP) that performs the binary classification task of deciding whether each pair $(x_{\text{head}}, x_{\text{dep}})$ in the Cartesian product of the input elements, i.e., $\{(\text{head}, \text{dep}) \mid \forall\ \text{head}, \text{dep} \in [1, \dots, \lambda]\}$, should be connected by a dependency arc. Depending on the input representation and the specific task we are targeting, there may be some pairs that are not connectable by a dependency arc, for example, pairs where head $=$ dep. For the GTTM input, pairs for which at least one element is a rest are also not connectable. Therefore, the binary classification is performed only on a subset of all pairs Λ that we call *potential arcs*. For every potential arc $(x_{\text{dep}}, x_{\text{head}})$, we predict the probability $\hat{y}_{\text{dep,head}}$ of a dependency arc by concatenating the two rows of $\mathcal{H}$ that correspond to the head's and the dependent's index into a single vector of length $2h$ and giving it as input to the MLP. We concatenate the two inputs instead of summing or multiplying them because our arcs are directed, so we need to preserve the order when aggregating the two embeddings. Moreover, despite the bilinear layer being a major selling point of Dozat's paper [26], we find that the concatenation approach yields better results. We can collect the output for all potential arcs into a *weighted graph-adjacency matrix* $\hat{Y}$, which is a $\lambda \times \lambda$ matrix with entries $\hat{y}_{\text{head,dep}}$ at the corresponding indices. We assign a probability 0 to the matrix entries that correspond to arcs $\notin \Lambda$.

4.3 Training Loss

In the training phase, we use the sum of the binary-cross-entropy (BCE) loss and the (multiclass) cross-entropy (CE) loss. The BCE loss is computed independently for each potential arc and measures the difference between the ground-truth label (0 or 1) and the predicted probability. We also use the CE loss because our problem can be framed as a multiclass classification problem where for each element we predict his head among $\lambda + 1$ possibilities (each sequence element plus a dummy element for the root and rests elements). The CE loss is therefore applied column-wise to the adjacency matrix $\hat{Y}$ predicted by our model.

In NLP, the BCE loss was used by [27] while the CE loss is used more generally, for example, by [26, 39]. We experimentally found that the sum of the two losses yields the best results.

4.4 Postprocessing

Since the prediction of our model is made independently for each potential arc, simply taking the row-wise maximum of the weighted adjacency matrix to select which head to assign to each element of the sequence could produce dependency cycles and, therefore, not yield a tree structure. We use a maximum-spanning-tree algorithm to find the tree over $\hat{Y}$ with the highest weight. Since our dependency trees are projective, we use the Eisner algorithm [40] which solves this problem using bottom–up dynamic programming with a time complexity of $O(\lambda^3)$. For applications involving non-projective trees other post-processing approaches such as Chu-Liu/Edmonds [41, 42] ($O(\lambda^2)$) are implemented in our framework.

5. EXPERIMENTS

Below, we describe the datasets, evaluation metrics, and experimental settings for the two kinds of trees we consider.

5.1 Datasets and preprocessing

We obtain the melodic time-span trees from the GTTM database [17], which contains MusicXML encodings of monophonic scores and a dedicated XML-based encoding of the constituent time-span trees (among other trees that we don't consider in this paper). We extract the note features with the Python library Partitura [43]. Some pieces have two different trees, and we keep only the first. We also discard 4 pieces that we could not import due to inconsistencies in the XML file encoding. In total, we have 296 melodies of lengths between 10 and 127 (notes + rests). For training, we augment the dataset by considering all transpositions between one octave higher and one octave lower.

We obtain the chord analyses from the Jazz Harmony Treebank (JHT) [9], which encodes both chord labels and harmonic analyses as constituent trees, in JSON format. As discussed in Section 3 this format does not distinguish between the primary and the secondary child when both have the same chord label. In this case, we assume by default that the right is the primary. The dataset contains two kinds of trees: open and complete constituents. We use the former for comparison reasons since the results are reported only for those [20]. In total, we have 150 sequences of lengths between 11 and 38 chords. For training, we augment the dataset by considering all 12 possible transpositions of the chord roots.

5.2 Evaluation metrics

The papers we compare use different metrics, and we implement all of them. The work of Harasim [20] uses two metrics, one more relevant for dependency trees and the other for constituent trees. The first is the *arc accuracy*, i.e., the normalised cardinality of the intersection between the set of predicted arcs and the ground truth arcs. The second is the *span accuracy*, computed as the normalised cardinality of the intersection between all the spans of the predicted constituent tree (i.e., the pair of the leftmost and rightmost leaf that is part of the subtree rooted at any non-leaf node) and the spans of the ground truth tree (see [20] for a more detailed explanation). Nakamura et al. [2] use the *node accuracy* metric, i.e., the normalised cardinality of the intersection between nodes in the predicted and ground truth trees, where two nodes are considered equal if the labels of the parent and children (or a dummy label if the node have no parent or children) are equal.

We also report another metric, the *head accuracy*, computed as the multiclass classification accuracy on the indices of the predicted heads, ordered by their dependent. For example for the dependency tree of Figure 1, this would correspond to the accuracy computed on the sequence $[4, 2, 3, 4, -1]$, where -1 indicates the root (which has no head). This is similar to the arc accuracy but enforces the presence of a dependency head for each sequence element (which may not be the case for a generic system), and gives more weight to the correct root prediction. It is also faster to compute and commonly used in NLP, so we include it to set a metric for future research. Note that all metrics presented above don't consider the nodes corresponding to rests, since they are only part of the input sequence, but not part of the tree.

5.3 Results

For our experiments, we set the hyperparameters of our encoder to an embedding size of 96, and 2 transformer layers of hidden size 64. The arc predictor (MLP) has 2 linear layers with the same hidden size. We use the GeLU activation [44] and the AdamW optimiser [45], with a learning rate of 0.0004 and weight decay of 0.05. We train with learning rate warm-up [46] of 50 steps and cosine annealing to limit the problem of high variance in the initial and final stages of training. The latter was particularly important

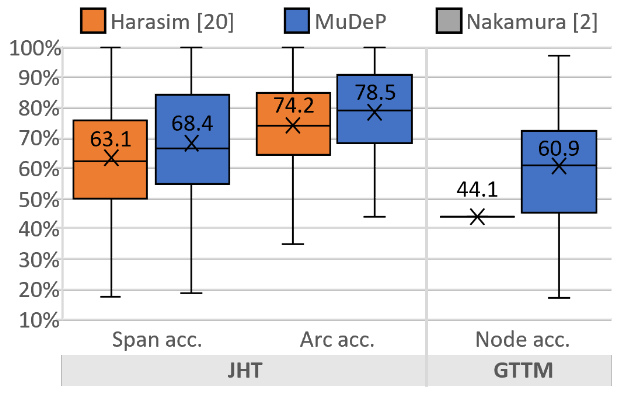


Figure 5. Boxplots of three accuracy metrics (higher is better) computed with leave-one-out cross-validation and their average. For Nakamura et al. [2], we report the average from their paper, so there is no deviation information.

since we did not use a validation set to perform early stopping due to the small size of our datasets. We train for 60 and 20 epochs for the JHT and GTTM datasets, respectively, since the latter is bigger and the data augmentation yields twice as many pieces in total). The training time is roughly the same, around 1 hour on a GPU RTX 1080.

We compare the results of our MuDeP on the JHT with Harasim [20] and on the GTTM with Nakamura et al. [2]. We use leave-one-out cross-validation, i.e., for a dataset with N pieces, we run our system N times, by training on $N-1$ pieces and evaluating with the remaining one. As shown in Figure 5, MuDeP outperforms previous methods.

By comparing the head accuracy between JHT and GTTM (79.2% vs 57.9%), it is clear that time-span prediction is a much harder problem than the chord analysis problem, despite the dataset being bigger. Another interesting result is that the span accuracy is lower than head accuracy for the JHT dataset (63.1%), but higher for the GTTM(64.8%). Apparently, the main problem for JHT is to select which two chords to connect, but the arc direction (i.e., which is the head and which is the dependent) is almost always correctly inferred; conversely, for the GTTM dataset, the system often connects the correct notes, but in the wrong direction. And this type of misprediction is punished in the head accuracy, but not in the span accuracy.

The full piece-wise statistics on all metrics, a graphical rendering of all our predicted trees, and the qualitative evaluation of some examples are available in our repository.

5.4 Ablation study

We report the difference in head accuracy averaged over 10 runs with 90/10 random train/test split for the JHT dataset. Regarding the loss, sole usage of the (multiclass) CE loss reduced the accuracy by 0.3%, and only using the binary CE loss reduced the accuracy by 4.1%. The use of a bilinear layer in the decoder reduced the accuracy by 1.2%. The absence of post-processing did not reduce the accuracy (when the network is fully trained, otherwise the reduction is very evident). This is promising but it does not automatically implies that the network is producing correctly formed trees since dependency loops could still be present in the output. There are also cases when the postprocessing is reducing the accuracy, by incorrectly deciding which arc to remove in a dependency loop.

6. CONCLUSION AND FUTURE WORK

We presented MuDeP, a system for the dependency parsing of music sequences, and a procedure to make it applicable to constituent trees. MuDeP improves upon previous methods, by incorporating the ability to consider multiple musical features simultaneously, taking advantage of sequential context, and handling noisy inputs robustly. Moreover, since it is based on widely researched deep learning components, it has the potential to scale to large datasets and longer sequences. The bottleneck for such scalability is the postprocessing algorithm with cubic complexity. Two solutions exist to this problem: if one is interested in non-projective trees, algorithms with a square complexity are available. Apart from that, our system is already having good accuracy without the postprocessing phase, as highlighted in the ablation study. Therefore, a faster heuristic may suffice to correct the few problematic dependencies without decreasing the performance.

Since our deep learning model is a black box, it is notably complicated to find a human-understandable explanation of its functioning. Although work in this direction exists [47, 48], it is still very limited [49]. Therefore, our model is mainly intended for scenarios in which one is only interested in obtaining the parsing trees, for example, to use them as input for another MIR task. Conversely, this paper might have limited utility if one's goal is to model music understanding and interpretation by humans. Grammar-based models are much more suitable for this goal, although there is a (somewhat speculative) possibility that the dependency-arc probabilities in our approach relate to first-guess heuristics.

As research on the deep learning components we use is rapidly evolving, any new discovery is likely to benefit our system. Self-supervised pretraining on larger datasets of monophonic music or chord sequences, for example by predicting next or masked tokens, could also improve the performance, as already proved for language parsing. While the goal of this paper was to present a general framework, we can also think about several domain-specific improvements, for example, training the GTTM time-span parser with a multi-target approach to predict at the same time the metrical, time-span, and prolongation structure. We hope that this work will motivate the development of more datasets of hierarchical music analyses, including datasets of dependency trees, which may be a valid alternative to constituent structures, and even open up more possibilities due to the missing projectivity constraints. Finally, we intend to explore in future research how the knowledge encoded in our model could be reused to guide other tasks, for example, automatic chord recognition from audio files.

7. ACKNOWLEDGEMENTS

This work is supported by the European Research Council (ERC) under the EU's Horizon 2020 research & innovation programme, grant agreement No. 101019375 ("Whither Music?"), and the Federal State of Upper Austria (LIT AI Lab).

8. REFERENCES

[1] S. Abdallah, N. Gold, and A. Marsden, "Analysing symbolic music with probabilistic grammars," *Computational music analysis*, pp. 157–189, 2015.

[2] E. Nakamura, M. Hamanaka, K. Hirata, and K. Yoshii, "Tree-structured probabilistic model of monophonic written music based on the generative theory of tonal music," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 276–280.

[3] M. Hamanaka, K. Hirata, and S. Tojo, "Time-span tree leveled by duration of time-span," in *Proceedings of the International Symposium on Computer Music Multidisciplinary Research (CMMR)*, 2021, pp. 155–164.

[4] C. Finkensiep and M. A. Rohrmeier, "Modeling and inferring proto-voice structure in free polyphony," in *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, 2021, pp. 189–196.

[5] M. Rohrmeier, "Towards a generative syntax of tonal harmony," *Journal of Mathematics and Music*, vol. 5, no. 1, pp. 35–53, 2011.

[6] M. Granroth-Wilding and M. Steedman, "A robust parser-interpreter for jazz chord sequences," *Journal of New Music Research*, vol. 43, no. 4, pp. 355–374, 2014.

[7] D. Harasim, M. Rohrmeier, and T. J. O'Donnell, "A generalized parsing framework for generative models of harmonic syntax." in *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, 2018, pp. 152–159.

[8] O. Melkonian, "Music as language: putting probabilistic temporal graph grammars to good use," in *Proceedings of the ACM SIGPLAN International Workshop on Functional Art, Music, Modeling, and Design*, 2019, pp. 1–10.

[9] D. Harasim, T. J. O'Donnell, and M. A. Rohrmeier, "Harmonic syntax in time: rhythm improves grammatical models of harmony," in *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, 2019, pp. 335–342.

[10] F. Foscarin, F. Jacquemard, and P. Rigaux, "Modeling and learning rhythm structure," in *Proceedings of the Sound and Music Computing Conference (SMC)*, 2019.

[11] F. Foscarin, F. Jacquemard, P. Rigaux, and M. Sakai, "A parse-based framework for coupled rhythm quantization and score structuring," in *Proceedings of the Mathematics and Computation in Music International Conference (MCM)*. Springer, 2019, pp. 248–260.

[12] F. Foscarin, R. Fournier-S'Niehotta, and F. Jacquemard, "A diff procedure for xml music score files," in *Proceedings of the International Conference on Digital Libraries for Musicology (DLfM)*, 2019.

[13] M. Rohrmeier, "Towards a formalization of musical rhythm." in *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, 2020, pp. 621–629.

[14] W. T. Fitch and M. D. Martins, "Hierarchical processing in music, language, and action: Lashley revisited," *Annals of the New York Academy of Sciences*, vol. 1316, no. 1, pp. 87–104, 2014.

[15] M. Tsuchiya, K. Ochiai, H. Kameoka, and S. Sagayama, "Probabilistic model of two-dimensional rhythm tree structure representation for automatic transcription of polyphonic midi signals," in *Proceedings of the Asia-Pacific Signal and Information Processing Association Annual Summit and Conference*. IEEE, 2013, pp. 1–6.

[16] I. Sakai, "Syntax in universal translation," in *Proceedings of the International Conference on Machine Translation and Applied Language Analysis*, 1961.

[17] M. Hamanaka, K. Hirata, and S. Tojo, "Musical structural analysis database based on gttm," in *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, 2014, pp. 325–330.

[18] D. Harasim, C. Finkensiep, P. Ericson, T. J. O'Donnell, and M. Rohrmeier, "The jazz harmony treebank," in *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, 2020, pp. 207–215.

[19] M. Rohrmeier, "The syntax of jazz harmony: diatonic tonality, phrase structure, and form," *Music Theory and Analysis (MTA)*, vol. 7, no. 1, pp. 1–63, 2020.

[20] D. Harasim, "The learnability of the grammar of jazz: Bayesian inference of hierarchical structures in harmony," Ph.D. dissertation, EPFL, 2020.

[21] C. Finkensiep, "The structure of free polyphony," Ph.D. dissertation, EPFL, 2023.

[22] M. Hamanaka, K. Hirata, and S. Tojo, "Implementing "a generative theory of tonal music"," *Journal of New Music Research*, vol. 35, no. 4, pp. 249–277, 2006.

[23] ——, "FATTA: Full automatic time-span tree analyzer," in *International Computer Music Conference (ICMC)*, vol. 1, 2007, pp. 153–156.

[24] ——, "deepGTTM-III: Multi-task Learning with Grouping and Metrical Structures," in *Proceedings of the International Symposium on Computer Music Multidisciplinary Research (CMMR)*, 2018, pp. 238–251.

[25] Y.-R. Lai and A. W.-Y. Su, "Deep learning based detection of GPR6 GTTM global feature rule of music scores," in *Proceedings of the International Conference on New Music Concepts*, vol. 56, 2021.

[26] T. Dozat and C. D. Manning, "Deep biaffine attention for neural dependency parsing," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[27] ——, "Simpler but more accurate semantic dependency parsing," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2018.

[28] X. Wang, J. Huang, and K. Tu, "Second-order semantic dependency parsing with end-to-end neural networks," 2019, pp. 4609—-4618.

[29] H. He and J. D. Choi, "Establishing strong baselines for the new decade: Sequence tagging, syntactic and semantic parsing with bert," in *Proceedings of the International Florida Artificial Intelligence Research Society Conference*, 2019, pp. 228–233.

[30] M. Zhang, "A survey of syntactic-semantic parsing based on constituent and dependency structures," *Science China Technological Sciences*, vol. 63, no. 10, pp. 1898–1920, 2020.

[31] F. Lerdahl and R. S. Jackendoff, *A generative theory of tonal music*. MIT press, 1985.

[32] J. Daniel, M. James H *et al.*, *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Prentice Hall series in artificial intelligence, 2007.

[33] L. Kong, A. M. Rush, and N. A. Smith, "Transforming dependencies into phrase structures," in *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2015, pp. 788–798.

[34] N. Fradet, J.-P. Briot, F. Chhel, A. El Fallah Seghrouchni, and N. Gutowski, "MidiTok: A python package for MIDI file tokenization," in *Late-Breaking Demo Session of the International Society for Music Information Retrieval Conference (ISMIR)*, 2021.

[35] N. Fradet, J.-P. Briot, F. Chhel, A. E. F. Seghrouchni, and N. Gutowski, "Byte pair encoding for symbolic music," *arXiv preprint arXiv:2301.11975*, 2023.

[36] EuroCC National Competence Center Sweden (ENCCS), "Graph neural networks and transformer workshop," https://enccs.github.io/gnn_transformers/notebooks/session_1/1b_vector_sums_vs_concatenation/, 2022.

[37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[38] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," in *Proceedings of the North American Chapter of the Association for Computational Linguistics*, 2018.

[39] D. Fernández-González and C. Gómez-Rodríguez, "Transition-based semantic dependency parsing with pointer networks," *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2020.

[40] J. M. Eisner, "Three new probabilistic models for dependency parsing: An exploration," in *Proceedings of the International Conference on Computational Linguistics (COLING)*, 1996.

[41] J. Edmonds, "Optimum branchings," *Journal of Research of the national Bureau of Standards*, vol. 71, no. 4, pp. 233–240, 1967.

[42] Y.-J. Chu, "On the shortest arborescence of a directed graph," *Scientia Sinica*, vol. 14, pp. 1396–1400, 1965.

[43] C. E. Cancino-Chacón, S. D. Peter, E. Karystinaios, F. Foscarin, M. Grachten, and G. Widmer, "Partitura: A Python Package for Symbolic Music Processing," in *Proceedings of the Music Encoding Conference (MEC)*, Halifax, Canada, 2022.

[44] D. Hendrycks and K. Gimpel, "Gaussian error linear units (GELUs)," *arXiv preprint arXiv:1606.08415*, 2016.

[45] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations (ICLR)*, 2019.

[46] X. S. Huang, F. Perez, J. Ba, and M. Volkovs, "Improving transformer optimization through better initialization," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2020, pp. 4475–4483.

[47] S. Mishra, B. L. Sturm, and S. Dixon, "Local Interpretable Model-agnostic Explanations for Music Content Analysis," in *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, 2017, pp. 537–543.

[48] F. Foscarin, K. Hoedt, V. Praher, A. Flexer, and G. Widmer, "Concept-based techniques for "musicologist-friendly" explanations in a deep music classifier," in *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, 2022.

[49] V. Praher, K. Prinz, A. Flexer, and G. Widmer, "On the veracity of local, model-agnostic explanations in audio classification: targeted investigations with adversarial examples," in *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, 2021.