Symbolic Music Alignment: Traditional and recent algorithms

Nina Braunmiller
student ID: k11923286
Johannes Kepler University Linz
braunmiller.nina@tutanota.com

Abstract

This work investigates different note alignment methods in symbolic music processing. Note alignment aims for keeping track of a musical performer. It shall convert played notes into other forms of abstract note representations, for example, score on a sheet of music. This work explains that it is still a hard task to solve. The work focuses on the aspect of graph-based alignment, meaning that graphical representations of music are involved in processing. Firstly, the more "traditional" method of Hidden Markov Models (HMM) is explained. It can be shown that this approach struggles with some issues, like stability. It is also discussed how the Dynamic Time Warping (DTW) algorithm can be used in a symbolic alignment task. Modern approaches of the last five years are the last reviewed topic. These approaches are complex and bring up innovative ideas of graph-based alignment by using machine learning. It becomes clear that there is still a massive lack of graph-based symbolic music alignment research regarding machine learning methods. Therefore, the underlying work comes to the conclusion that it is worth extending new techniques to the field of music alignment. Even further approaches which can handle noisy data robustly would be quite welcome.

1 Introduction

As will be seen in the upcoming thesis, research tackles the question of following a piece of music. More precisely, it focuses on technology which shall help humans to understand where in a current piece of music the performer is located at the moment. This becomes possible by marking the currently played scores in a sheet of music. The help of technology is absolutely needed because it would be very time-consuming when a human expert would mark every played note on a score beforehand in the case of performance-to-score alignment. With the help of algorithms, it shall be possible to track a piece of music automatically [6].
Following the track note by note with the help of symbolic input is called note alignment. So, symbolic-to-symbolic alignments are meant where "symbolic" stands for the symbolic representation of music information in the form of, for instance, MIDI or MusicXML score files. As becomes clear by fig. 1(left), the note alignment copes with slight time shifts and the fact that the input performance doesn't need to contain identical musical information as the targeted representation [6].
Fig. 1(right) also mentions sequence alignment. It aims to fit features of sequences together. A sequence can be an audio signal but can also refer to audio-to-score alignments because concatenated scores can be interpreted as sequence [6].

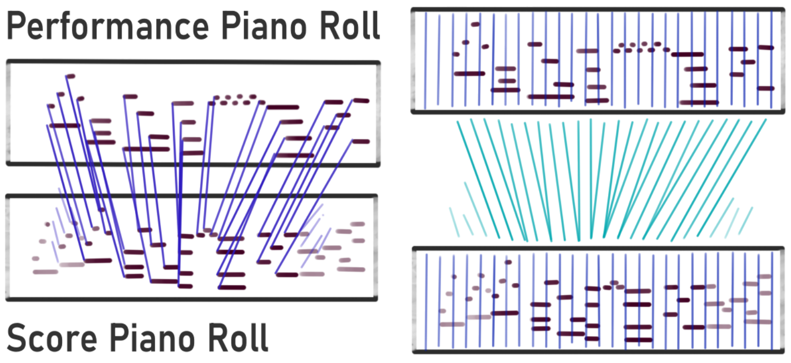


Figure 1: Draft of note alignment (left) and sequence alignment (right) [6].

The underlying work will discuss the most diverse graph-based music alignment strategies. The concluding chapter will shortly discuss the huge challenge of symbolic music alignment in modern research.

2 Studies concerning graph-based alignment

The following sub-chapters describe methods to build graphs used for symbolic music alignment. The papers concentrate on polyphonic music meaning that several notes can be played at the same time [7]. Additionally, real-time alignment shall become possible. Therefore, algorithmic speed is an important component.

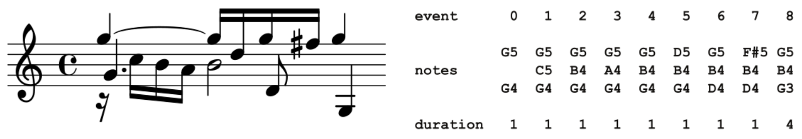
In the first sub-chapter, Hidden Markov Models (HMMs) are considered as traditional approaches as they are long-known in research. They use probabilistic theory in building up HMM graphs to make alignment possible. However, stability issues are not seldom [1, 5, 7].

It also makes sense to have a look at Dynamic Time Warping (DTW), an old but still used algorithm, for aligning sequences to sequences which can be seen as graphs [6, 8].

Last but not least, modern approaches using deep learning will be introduced to overcome the weaknesses of traditional approaches. As they introduce greater complexity, speed becomes a critical component. REpresentation learning-based Graph ALignment (REGAL) tries to overcome this issue by simply estimating instead of computing similarity [2].

2.1 Tracking scores with Hidden Markov Models (HMM)

As stated above this work considers first the HMM. The following three papers will further outline how HMM can be used in aligning inputs to scores.



event	0	1	2	3	4	5	6	7	8
	G5	G5	G5	G5	G5	D5	G5	F#5	G5
notes		C5	B4	A4	B4	B4	B4	B4	B4
	G4	G4	G4	G4	G4	G4	D4	D4	G3
duration	1	1	1	1	1	1	1	1	4

Figure 2: Visualizing a sample score (left) and its event sequence (right) [5].

Montecchio and Orio (2009) represent the scores and the audio data as HMM graphs for the purpose of audio-to-score alignment. Therefore, the paper is split into two parts.
First, it has a look at the score level. The belonging graph contains states that represent events. It claims that in a polyphonic setting, the events are constrained by one onset and one offset. Fig. 2 explains how the event can be defined technically. One chord carries several simultaneously played

notes. It becomes clear that an event always ends as soon as one note within the chord ends. The other notes within that chord don't have to end but will be part of the upcoming events [5].
The alignment can be negatively affected by incorrectly played or skipped notes. To hinder those effects to become a global alignment problem, ghost states are introduced. These are simply empty nodes. Each state gets its own accompanying ghost state. There is the option to transition to the ghost state or to the next usual state. When the ghost state is chosen a self-transition or a transition to further away states becomes possible. When jumping from a ghost state to a usual state a decreasing function is used to prefer close-by states as best candidates to hop on [5].

Coming to the audio signal, it is split into Fast Fourier Transform (FFT) bins where each represents a range of energy-weighted frequencies carrying the currently played chord. However, FFT leads to a loss of the lower frequency range. Therefore, discrete filters to manipulate the filter bandwidth can be used. The individual filters refer to different notes, like C3. The study aims to find out which current note is played when concerning score-related filters. Therefore, the probability of observing the i-th sustain state s is given by $b_i^s = similarityFunction\left(\frac{E_i^n}{EnergyAudioSnippet}\right)$. The E_i^n is the energy of the expected frequency band [5].

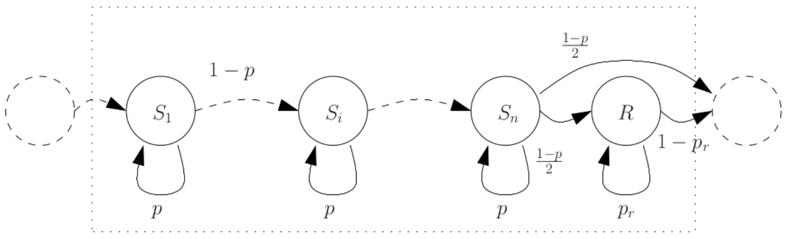


Figure 3: Event-level of score graph modeling where S describes the sustain state, R is the rest state, and p is used as the probability for state change [5].

Formulating the event level as an HMM graph, different kinds of states can be formulated. Sustain states are introduced which describe an ongoing event. As long the event goes on there is a probability to stay in that state p of to transition to the next state 1-p. At the end of such a sustain chain, a rest state may appear which describes a pause after an event. It can be expanded by a self-loop. Fig. 3 describes the probabilities of such a chain more precisely [5].

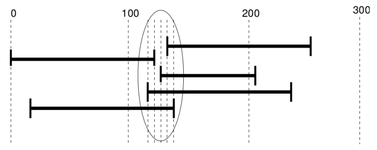


Figure 4: Precision lack in chord design through small time shifts of single notes visualized by bars [7].

The filter banks of Montecchio and Orio (2009) could be used to find out the played notes needed for MIDI creation and alignment. Therefore, the study could have implemented MIDI before alignment. When directly using MIDI-to-score alignment, note alignment is given [6]. Schwarz et al. (2004) investigate it more closely by following MIDI files online in a polyphonic structure.

Also in Schwarz et al. (2004), the MIDI model and the score model have to be developed separately. For the latter, three possible states are assumed. The polyphonic state is for several notes at once. The monophonic state is only active as long as one note at once is played. The rest state is alive when music is silenced. Like in the work of Montecchio and Orio (2009), ghost states are used to overcome the problem of skipped, wrong, and added events for score representation [7].

Furthermore, it is assumed that notes are not perfectly synchronized as there may be slight shifts of notes that appear like it is presented in fig. 4. Shifts are also assumed for the legato notes which should be sub-sequentially played when one note ends and the next note exactly starts in theory. The paper avoids too heavy state building through shifts by fixing a window lower boundary of 30 ms as one event. Additionally, the breaks, which are shorter than 100 ms, are removed [7].

Also, the MIDI has to be encoded. First of all, it is pointed out that there may be notes active that have been volumeless for a while. The pedal activation causes this phenomenon. Therefore, it is assumed that a note decays with time for every note in the same manner. The time measurement in MIDI is the unit thick. Through that, it becomes possible to compute the note match given by

$$m_n = \begin{cases} \frac{EnergyAllExpectedNotes}{EnergyAllCurrentMIDINotes} \end{cases}$$

when the energy in MIDI is given [7].

The HMM states are conceptually like the ones of Montecchio and Orio (2009). Schwarz et al. (2004) formulate an attack state followed by a sustain state of the note. When the note ends it might lead to the release state representing the minimal break after a note when no legato is played. All of these states contain the probability of self-looping [7].

Training of transition probabilities, like the ones in fig. 3, was executed with a view on error robustness and speed recognition. Results show that the HMM struggles with the difference between expected and actual tempo [7]. Gu and Raphael (2009) want to fix the issues with tempo estimation in the same polyphonic MIDI-to-score alignment setting. Instead of creating states out of an event, each note is individually encoded as a state with skip and self-loop options. When notes are simultaneously played, the model considers all possible sort options of those notes. k! orders for k notes [1].

The probability to enter the next state in HMM can be computed with $p(x_{n+1}|x_n) = q(L(x_{n+1}) - L(x_n))/B(x_{n+1})$. $B(x_{n+1})$ stands for the number of simultaneously played notes in the state of HMM note x_{n+1}. L() simply gives information on how far we go starting from state 0. For example, L(2) would say that two steps were made from state 0, therefore it would represent the number 2. It follows that $q(L(x_{n+1}) - L(x_n))$ is in the focus of interest. When several steps are made at once then q(value>1) stands for the probability of skipping several notes. q(1) is the probability of simply moving one note further. q(0) means that the performer adds a note, therefore in HMM a self-loop would happen [1].

The performed note y_{n+1} is aligned to the score x^*_{n+1} when its probability $p(x^*_{n+1}|y_1, ..., y_{n+1})$ exceeds a fixed threshold and x^*_{n+1} the first chosen candidate out of chord competitors without being used before. The results are influenced by the threshold. When it is too high notes can be missed easily but the incorrect scoring will be lowered [1].

Gu and Raphael (2009) aim to predict the upcoming score and is updating this prediction with every new incoming information by estimating the rhythm. Parameters can be learned by adjusting the mean and variance based on the maximum likelihood of the last model. The model can be formulated as

$$\begin{pmatrix} p_{k+1} \\ s_{k+1} \end{pmatrix} = \begin{pmatrix} 1 & l_k \\ 0 & 1 \end{pmatrix} \begin{pmatrix} p_k \\ s_k \end{pmatrix} + \begin{pmatrix} \lambda_k \\ \sigma_k \end{pmatrix}$$

where p_k describes the time of the event, s_k the tempo, l_k the length of the event. λ and σ are independent, normal distributed vectors of mean 0. Learning them shall help to better time the score note to the referring performance note. Through training, fewer notes are missed and the latency of note tracking is lowered. Correct detections reached 88.7 % for the trained version and 84.8 % for the untrained case [1].

2.2 Symbolic music alignment with Dynamic Time Warping (DTW)

The Dynamic Time Warping (DTW) algorithm is a classic alignment algorithm that is reused and further developed nowadays [6]. As it aligns graphs to graphs it is well suited for this thesis. However, in research, there are lots of sequence alignment papers instead of symbolic-to-symbolic alignments. Nevertheless, Tsai et al. (2020) capture the idea of DTW. The authors work with photographed pieces of music sheets and MIDI files. Both have to be converted into a space that can be used as a graph-like structure to make use of DTW. Therefore, the paper focuses on converting symbolic music into such a space, namely the bootleg score. It is a hybrid structure of MIDI and sheet music. It is sparse and binary which is its huge computational advantage. To give an idea of the sparsity of needed information fig. 5 is provided. The assumption of Tsai et al. (2020) is that it is enough to only use filled noteheads, staff lines, and bar lines for alignment. Subsequence DTW is a modification of DTW developed to align short sequences with subsequences of a longer sequence. Therefore, it matches perfectly the task [8].

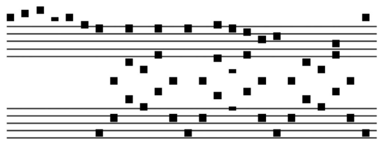


Figure 5: Bootleg score visualization with decorating lines which aren't part of the representation [8].

With that strategy of filtering the little information out of MIDI and music sheet photographs the run time is on average 0.9 seconds. This is much faster than other alignment competitors listed in the paper of Tasi et al. (2020). In addition, on average, only 808 bytes of computation power are needed. A F-measure of 0.89 is reached which is the best among the competitors [8].

2.3 Recent ideas of graph-based alignments: deep learning

Modern research with a focus on graph-based alignment brings up the idea of using deep learning methods to generate graph representations that align similar graphs [2, 4]. However, the question is still open on how to transfer data into appropriate graph representations. There seems to be no state-of-the-art approach.

Jeong et al. (2019) deal with the question of transferring scores into a graph-like structure. To make things happen the encapsulating architecture is the Variational Auto-Encoder (VAE) which takes a score MusicXML file as input. The output shall be a MIDI representation Y of the input. The VAE works as usual. The input gets in dimension compressed such that it ends up as a hidden embedded representation (encoding). Then with the help of a normal distribution samples are taken from that space with the aim of generating a MIDI file format with the same underlying input content (decoding). That means that the weights in the encoding and decoding space differ from each other. VAE learns through a regularized loss function which concerns encoding and decoding parts. The complete architecture is shown in fig. 6. The following text roughly explains what's happening in the figure [3].

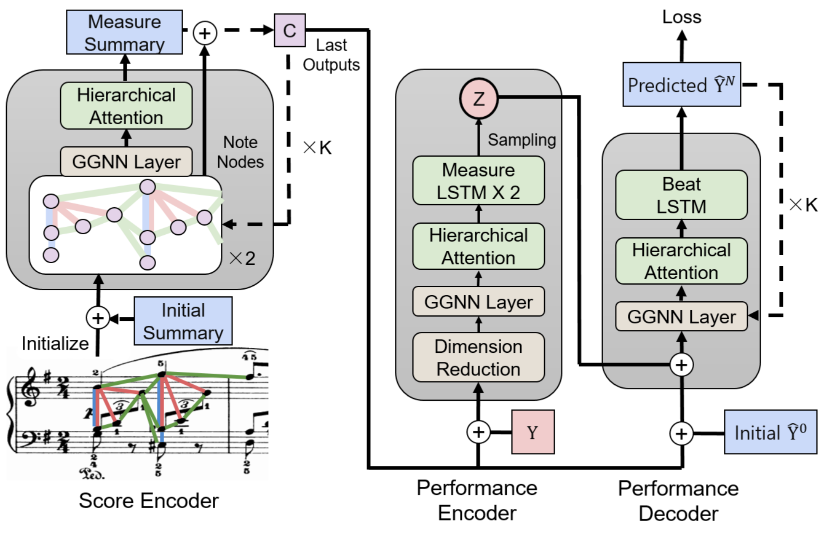


Figure 6: Architecture of the VAE where Y is the aimed MIDI output and the performance encoder is only used during training [3].

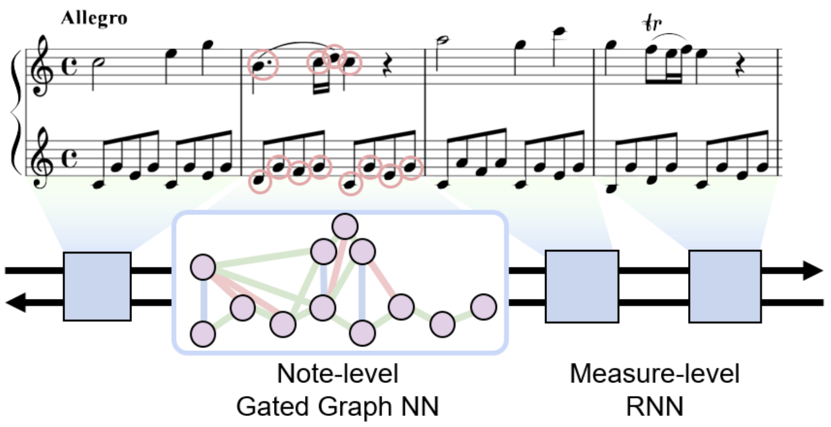


Figure 7: Graph design for the GGNN surrounded by context information tracker RNN (blue boxes) [3].

First of all, the MusicXML file has to be represented as a graph. Each note is transferred into a node. Fig. 7 gives the concept idea. The relationships between the notes are considered by edge encoding which can carry 12 different inter-note relation types [3].

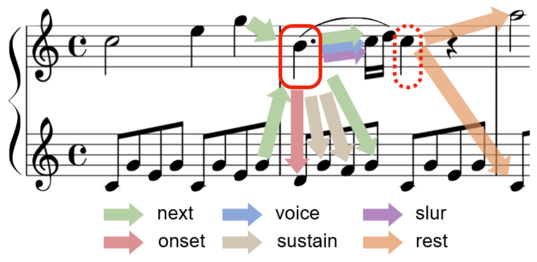


Figure 8: Edge types for graph design with the centering note (red box) [3].

Fig. 8 marks a note which also can be represented as a node by a red box. All its relationships to other nearby notes are given by directed darts which represent the edges and their possible types. The dart always points to a future note. Onset edges refer to simultaneously beginning notes. They are the only ones who are undirected. Next edges point to the following note when no break lies in-between. Voice edge is a next edge within the same voice. Notes which have a break in-between are connected by a rest edge. Notes connected by an arc get the slur edge. When a note hasn't ended yet but in chord there appear other notes, the dart points towards them marked as sustain edges. The missing pointer in fig. 8 is the one which points to the current note itself. Each type of edge gets its own weight parameter to learn. The above described graph is inputted in the form of an adjacency matrix into the Gated Graph Neural Network (GGNN) to enable the learning process. The "Gated" is used to describe the learning process as a Gate Recurrent Unit (GRU) is used. It outputs a hidden state representation of a note [3].

Beyond that, the paper points out that music is constructed in a hierarchical structure, like beat, measure, phrase, and section. Therefore, Hierarchical Attention RNN (HAN) was introduced as it can be used in the context of GNNs. It makes it possible to store all previous hidden states in the current hidden state such that sequences can be memorized. Attention is used to store only the important parts. The idea is to use attention heads in such a way that the context vector is trainable. Which context parts are important is learned by the network [3].

Combining GGNN and HAN results in the Iterative Sequential Graph Network (ISGN). More precisely, that means that GGNN's resulting graph is fed into HAN which takes care of the individual nodes by paying different attention levels to them. This contextualized graph is next analyzed by a Long Short-Term Memory (LSTM) which is a neural network working well for sequences. It also spends attention but in a wider sequence context [3].

ISGN is compared with models created out of HAN and a model only looking at note-wise LSTM and not at the context. It becomes clear that ISGN's produced MIDI files aren't significantly outperforming the others but reached according to the listening participants in the MIDI quality slightly better scores [3].

Jeong et al. (2019) use the performance encoder for MIDI only for training. However, it could be also used to encode MIDI files into a hidden representation space which also accounts for the score encoder. Through that, MIDI and score representations would become comparable and therefore alignable as will be seen in the following discussed paper. Even further, Jeong et al. (2019) give an example of how a music sheet can be encoded into a trainable graph structure. This is necessarily needed for the upcoming papers as they don't describe symbolic music representation [4, 5].

Li et al. (2019) make clear that Graph Neural Networks (GNN) evaluate the similarity of graphs by embedding them into vector space. However, the authors are convinced that it can reach even better performances by aligning each node individually with the other graph's nodes. This approach is called Graph Matching Network (GMN) [4].

The assumption is that the GNN converts graphs into a vector space where similar graphs lay closer together. When a graph shall be embedded into vector space the GNN architecture is as follows. It starts with several encoders that are alone-standing Multilayer Perceptrons (MLP). Two take features of two nodes as input, in the other features of one edge are inputted. Then the propagation layers follow. Here a MLP takes the three outputs of the layer in front, namely the outputs for two nodes and their belonging edge, to formulate a message. For the target node, all messages pointing towards it are computed. Its sum and the hidden node representation are then fed into another machine learning network like MLP/RNN/GRU/LSTM to get the newest node representation. The process is iterated where the latest current node and its recent neighbor node representations are inputted to formulate messages and even newer presentations. In such a way the aggregation of neighborhood information shall be concentrated in one node representation. In the last step, the aggregator takes all those new representations of a node i, $h_i^{(T)}$. It aims for computing a graph representation for all graph-contained nodes. For this reason, over all nodes i it is summed $\sigma(MLP_{gate}(h_i^{(T)})) \odot MLP(h_i^{(T)})$. This sum is again inputted into a MLP. The output will be the final graph representation [4].

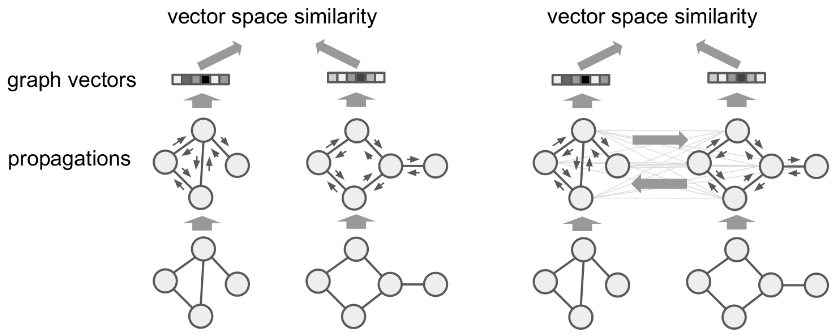


Figure 9: GNN (left) and GMN (right) [4].

Li et al. (2019) extend GNN by GMN. The slight difference is visualized by fig. 9. GMN takes two graphs as input and evaluates how similar these are. An attention mechanism is used for it. This shall be even better than the embedding model of GNN. GMN needs some more complexity. The message is computed identically to GNN. Cross-graph matching compares the nodes of two graphs by their similarity. This matching is an attention-based function that takes two encoded node representations from two graphs as inputs. More in detail, attention is defined by $a_{j->i} = \frac{exp(s_h(h_i^{(t}, h_j^{(t)})}{\sum_{j'} exp(s_h(h_i^{(t}, h_j'^{(t)})}$ where s_h stands for the vector space similarity as it was computed in GNN. As index i is related to the first graph and j refers to the second one, attention concerns the similarity between two nodes from different graphs. This attention is used for the matching function $\mu_{j->i} = a_{j->i}(h_i^{(t)} - h_j^{(t)})$. The propagation layer's machine learning network takes in addition to the same inputs as in GNN all matching functions $\mu_{...->i}$ pointing towards the current node i summed up as input. Finally, the aggregation step of GNN is done. GMN outperforms different competitors, like the GNN [4].

However, comparing every node with every other graph's node is quite complex. Such a process needs time and can be therefore too inefficient for real-time alignment. Therefore, Heimann et al. (2018) develop a strategy that only estimates the similarity matrix. Its heavy computation is avoided. The process path is explained in the upcoming text more closely.

Heimann et al. (2018) present the REpresentation learning-based Graph ALignment (REGAL) which shall automatically learn graph node representations and semantically connects nodes between two graphs.

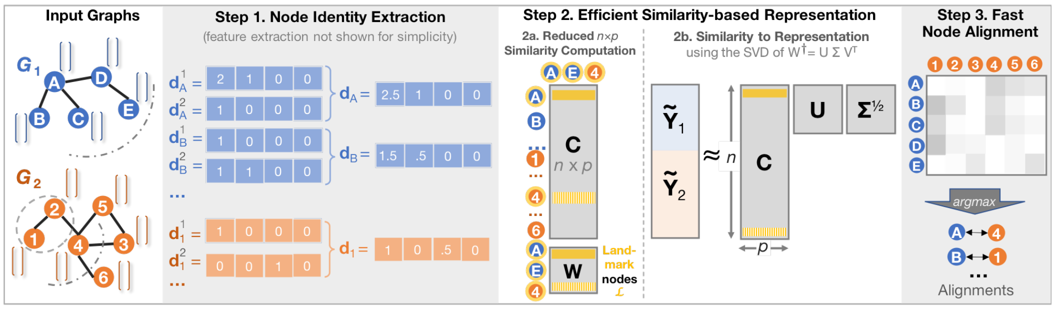

Input Graphs
G1
G2
Step 1. Node Identity Extraction
(feature extraction not shown for simplicity)
Step 2. Efficient Similarity-based Representation
2a. Reduced n×p Similarity Computation
2b. Similarity to Representation using the SVD of W† = U Σ V^T
Step 3. Fast Node Alignment
C n x p
W
Land-mark nodes
U
Σ½
argmax
Alignments

Figure 10: Whole procedure of REGAL where steps 1 and 2 create an estimated embedding (xNetMF) and step 3 aligns the single graph nodes through of previous steps [2].

Heimann et al. (2018) define unsupervised graph alignment as the process of learning and fitting node representations. Those shall generalize several graphs. Several graphs can be embedded and then compared due to similarity [2].

Fig. 10 visualizes the REGAL architecture. Graphs are embedded with the help of Cross-Network Matrix Factorization (xNetMF) which is defined as a representation learning method. In xNetMF node attributes and structural similarities are considered in the aimed embedding. However, proximity similarities are neglected. xNetMF builds up on a similarity matrix as matrix factorization. However, the costs of building such a matrix shall be kept low. The method is explained more in detail in the upcoming lines [2].

Parts of the xNetMF process are node identity extraction and efficient similarity-based representation. The first one can be described as follows. For every node, the attribute- and structure-relevant information is extracted. It is problematic when several graphs have no direct link to each other. Therefore, proximity approaches don't fit. Instead Heimann et al. (2018) look at the structural identity. Here the degree of the neighborhood is of interest. The neighborhood is on a border of k hops from the original node. The i-th entry of the vector carrying the relevant information is defined by $\lfloor log_2(degree(neighborhoodNode)) \rfloor = i$. With this method, the vector size shall be kept in adequate size. This can be done for all hop values <= K. Fig. 10 outlines that process. In step 1 those vectors are described as $d_{node}^{hopNumber}$. These $d_{node}^{hopNumber}$ can be merged to a vector containing all hops by $d_u = \sum_{k=1}^{K} \delta^{k-1} d_u^k$ where δ is a decay factor smaller than 1 giving more weight to the nearby neighborhood reachable with fewer hops [2].

Voluntarily, the attribute-based Identity can be used to make the performance even better. For each node, a vector f is built containing all node attributes/features. Edge attributes are ignored but could be easily converted into node attributes [2].

Bringing these two identity approaches together into a similarity measure for two nodes of two graphs, the paper ends up with the formula $sim(u, v) = exp[-\gamma_s \cdot ||d_u - d_v||_2^2 - \gamma_a \cdot distance(f_u, f_v)]$ where the first inner exp part refers to structural identity and the second to attribute identity. Both parts get scaled by γ [2].

The next xNetMF stage is the efficient similarity-based representation. Out of the identities, a similarity matrix S is built with the help of the above-stated similarity function. The aim is to compute the embedding without explicitly computing S [2].

To save computational power and to get noise stability, the first graph's nodes are only compared to a part of the second graph's nodes such that p « n. n is specified as the maximal number of the second graph's nodes. S can then be approximated with $\hat{S} = CW^{\dagger}C^{T}$ where C is shaped with n x p and W is a p x p matrix which gives the similarity of randomly chosen end comparison nodes. † marks a pseudo-inverse of a matrix. However, even $\hat{S}$ is created explicitly. Following steps make computation much cheaper.

$$U, \Sigma, V = SingularValueDecomposition(W^{\dagger})$$

$$\hat{Y} = CU\Sigma^{-0.5}$$

$$\hat{Y} = Normalize(\hat{Y})$$

$$\hat{Y}_{graph1}, \hat{Y}_{graph2} = split(\hat{Y})$$

where $\hat{Y}$ give the estimated version of embedding matrix Y [2].

The last step of REGAL is the fast node representation alignment. It finds in a greedy way the top-alpha similar embeddings between graphs. For soft alignment, it is suited to get the top alpha fitting nodes of the second graph matching one node in the first graph. For this purpose, the second graph embedding is stored in a k-d tree for faster search. There, the best alignment can be taken. For music alignment, it makes sense to only use the best alignment node which can be found via Euclidean distance between nodes of the two graph embeddings [2].

For accuracy and noisy partition of data lower than 3 %, REGAL outperformed all its competitors which are network alignment methods. It ranged between accuracies of circa 65 and 85 %. However, when the noise level gets higher, the REGAL version with the DTW-using embedding method struc2vec instead of xNetMF supplies the best performance. On the other hand, REGAL is much faster than REGAL with the DTW-complex struc2vec [2].

It becomes clear that REGAL performs quite well but its huge weak point is its drastic influence by noise. Therefore, the two REGAL versions seem to be a success. However, for real-time music alignment, there should be a low noise rate provided [2].

3 Conclusion

The underlying work is only a little jump into the field of symbolic music alignment. Literature research made clear that this field seems to be a sneeze which isn't really considered in research. There is a massive lack of newer innovative approaches as Heimann et al. (2018) and Li et al. (2019) are focused on graphs but not on music. These are one of the very few fitting papers related to the seminar work's topic. Therefore, it would make sense to increase innovative research efforts in this field.

References

[1] Gu, Y. & Raphael, C. (2009) Orchestral Accompaniment for a Reproducing Piano. In *Proceeding of the International Computer Music Conference (ICMC09)*, pp. 501–504. Montreal, Canada. url: `https://quod.lib.umich.edu/i/icmc/bbp2372.2009.113/1`

[2] Heimann, M., Shen, H., Safavi, T. & Koutra, D. (2018) REGAL: Representation Learning-based Graph Alignment. In *ACM International Conference on Information and Knowledge Management 27*, pp. 117–126. New York, NY: Association for Computing Machinery. doi: 10.1145/3269206.3271788

[3] Jeong, D., Kwon, T., Kim, Y. & Nam, J. (2019) Graph Neural Network for Music Score Data and Modeling Expressive Piano Performance. In *International Conference on Machine Learning 36*. url: `http://proceedings.mlr.press/v97/jeong19a/jeong19a.pdf`

[4] Li, Y., Gu, C., Dullien, T., Vinyals, O. & Kohli, P. (2019). Graph Matching Networks for Learning the Similarity of Graph Structured Objects. In *International Conference on Machine Learning 36*. doi: 10.48550/arXiv.1904.12787

[5] Montecchio, N. & Orio, N. (2009) A DISCRETE FILTER BANK APPROACH TO AUDIO TO SCORE MATCHING FOR POLYPHONIC MUSIC. In *International Society for Music Information Retrieval Conference 10*, pp. 495-500. url: `https://ismir2009.ismir.net/proceedings/PS3-18.pdf`

[6] Peter, S., Cancino-Chacón, C.E., Karystinaios, E., Foscarin, F., McLeod, A. & Widmer, G. (in press) Automatic Note-Level Score-To-Performance Alignments in the ASAP Dataset. *Transactions of the International Society for Music Information Retrieval (TISMIR)*.

[7] Schwarz, D., Orio, N. & Schnell, N. (2004) Robust Polyphonic Midi Score Following with Hidden Markov Models. In *Proceedings of the International Computer Music Conference (ICMC)*, pp. 1–4. Miami, FL. url: `https://quod.lib.umich.edu/i/icmc/bbp2372.2004.061/1`

[8] Tsai, T.J., Yang, D., Shan, M., Tanprasert, T. & Jenrungrot (2020) Using Cell Phone Pictures of Sheet Music To Retrieve MIDI Passages. *IEEE Transactions on Multimedia* **22**(12). doi: 10.1109/TMM.2020.2973831