Unsupervised Harmonic Sound Source Separation with Spectral Clustering

Yiming Lin * yl4853@columbia.edu
Lucy Wang * kw2960@columbia.edu
Lingyu Zhang * lz2814@columbia.edu
Zhaoyuan Deng * zd2286@columbia.edu

Abstract

With the growing amount of audio data generated and processed in modern day media industry, music retrieval largely depends on automated algorithms. Separation of harmonic music components plays a crucial role in labeling and retrieving music data. Due to the amount of labeling required for instrument identification, supervised methods are often difficult to perform. In this work, we first implemented the unsupervised method proposed in [8]. We modeled mixed sources of audio signals by sinusoidal modeling with Short-Time Fourier Transforms. Based on selected spectral peaks of sinusoidal parameters, we constructed a similarity function between time and frequency components, and applied spectral clustering to globally partition the data. We then define a loss metric to evaluate clustering results and analyzed the effectiveness and weighted combinations of similarity features as well as clustering methods. We finally propose a fast implementation of Harmonically Wrapped Peak Similarity to run full graph spectral clustering, taking only 3.2% of the time of the naive implementation. Our code has been made available at https://github.com/Mikeyboiii/Spectral_Harmony.

1. Introduction

Separation of harmonic sound sources in a mixed sound signal has been a challenging problem. The main difficulty arises from the fact that musical instruments are almost always overlapping in time and frequencies, thus cannot be separated by naive time frame segmentation or frequency domain filtering. Over the years, many different approaches targeting specific areas of music source separation has been developed and achieved great results. The most prevalent method is to employ neural networks to learn to identify instruments in a supervised manner. These neural network based approaches can be further separated into two categories: spectrogram based or waveform based methods. Works like Open-Unmix [4], D3Net [14] use spectrogram either presented as amplitude or concatenation of its real and imaginary parts as input. Works like Wave-U-Net [13], Demucs [3] directly use raw waveform as input. Currently, the state-of-the-art model for this task is HyBrid Demucs [2], which uses a combination of spectrogram and waveform for separation, and has achieved a Signal-to-Distortion Ratio (SDR) of 7.68 dB. However, since the above methods are all supervised, they all require huge computational power and a large amount of training data. The types of instruments that can be separated are also influenced by the selection of training data, resulting in questionable ability to generalize to unseen types of instruments.

Unsupervised methods of music source separation have also been studied extensively in literature. One common approach to the task is Independent Component Analysis (ICA). However, most work that employ traditional ICA require a multi-channel mixture signal that is created by more than one microphone [5]. For many applications, multi-perspective observed data is not available. The need for separation usually happens in single channel mixes. While there exists single-channel methods, they usually have more limitations and require strong additional assumptions. For example, SCICA[1] only works for specific situations where the spectra of the signals to separate do not overlap in frequency, and it does not preserve the amplitude and phase of the initial signals. Other works [9] focus on utilizing sinusoidal parameters for local component matching. In order to consider global information for higher-quality separation, we move our attention to spectral clustering method with sinuous sound model which is described in [8].

As for the datasets used for music source separation, MUSDB18 dataset [12] is the most common benchmark dataset and has been extensively used in other researches. MUSDB18 contains 150 tracks with 4 instrument categories, which includes bass, drums, vocal, and others. Other popular datasets for music source separation include Slakh2100, a dataset of multi-track audio that contains 2100 automatically mixed tracks with 34 instrument categories, MedleyDB, a dataset that includes melody annotations and contains 122 songs with 82 instrument categories, and MIR1K which contains 1000 song clips and is designed

[1]* Fu Foundation School of Engineering and Applied Science, Columbia University

for singing voice separation. In this work, since benchmarks on MUSDB18 are often used by supervised methods, we only use it for algorithm design and visualization. We constructed a toy sample and introduced a loss function for evaluation, described in Section 3.

2. Methods

The separation process mainly consists of two steps. First, we need a representation of the data with small enough units of information that is able to disentangle the overlapping components. Second, we need to construct an effective similarity metric among the units and find partition them accordingly. In other words, the first step breaks the audio data into smaller units which can be assumed generated from only one source, the second step finds pattern between these units are assign them into groups. In this work, we apply sinusoidal modeling for audio data representation, construct similarity based on amplitude, frequency and harmonicity, and cluster units with spectral clustering.

2.1. Sinusoidal Modeling

The front-end representation of audio signals is of great importance in auditory scene analysis, and could sometimes be the bottleneck of performance. In our case, the goal is to obtain a quantized representation with the following properties. First, there should be minimal information loss. Manipulation of data in any way will cause some portion of the original data to be lost or damaged. It is ideal to preserve the information crucial to downstream tasks while sacrificing non-essential parts. Secondly, the representation should in some way be related to physical properties. Separating musical signals largely depends on the physical difference between each component, and it is the scientific basis of this task. Finally, we want a representation that is easy to play with and fits well with off-the-shelf clustering algorithms. [11] Considering the above requirements, we adopt the sinusoidal modeling method applied in [8]. Sinusoidal modeling in audio analysis was originally proposed for speech [10]. With Short-Time Fourier Transform (STFT), audio waves are decomposed into frames of sinusoidal, which are characterized by amplitude, frequency and phase. In specific, we model the discrete signal frame at time k with a sum of sinusoidal components with different amplitude, frequency and phases:

$$x_k(n) = \sum_{l=1}^{L_k} a_{lk} \cdot cos(\frac{2\pi}{F_s} f_{lk} \cdot n + \phi_{lk})$$

With F_s being the sample frequency, a_{lk} and ϕ_{lk} being the amplitude and phase of the lth frequency component of the kth frame, respectively. Unfortunately, a precise decomposition of an arbitrary signal requires infinite cosine components, which is intractable for both storing and computing. However, if the frames are short in time, they can be assumed stationary and have structures simple enough to be approximated by a manageable number of components, without severe information loss. In practice, the majority of frequency components in a frame have little contribution to the sum. To make it more data efficient, only the most prominent spectral peaks (components with the largest amplitudes) are selected for later process. To sum up, we first apply STFT to the input file, obtaining an (amplitude, frequency, phase) triplet as parameters for every frequency component in every frame. For every frame, we sort the frequency components by their amplitudes and select the top 20 prominent peaks to represent the data. In the end, for an audio file sampled into K frames and keeping the top $L = 20$ components, an L by K peak matrix X is computed, with the X_{lk} entry being an (amplitude, frequency, phase) triplet corresponding to the lth component in the kth frame.

2.2. Spectral Clustering

To separate the mixed audio signals, we want to find a partition among the selected peaks. In other words, we need to group similar frequency components across frames together. As proposed in [8], we first construct a similarity matrix of the peaks according to their amplitudes, frequencies and harmonicity. Then we apply spectral clustering to find a global normalized cut of the corresponding graph.

2.2.1 Similarity Matrix Construction

The first step of spectral clustering is to construct the similarity matrix from a graph of sound sources. In order to fully utilize the sound source information, we construct a graph using the entire duration of the sound mixture. According to [7], we first construct $K \cdot L$ nodes, each representing a peak in matrix X. Since it is difficult to find a physical interpretation of phase shift information, for each pair of of peaks $X_{lk} = \{a_{lk}, f_{lk}, \phi_{lk}\}$, $X'_{lk} = \{a'_{lk}, f'_{lk}, \phi'_{lk}\}$, we only use amplitude and frequency information, and define their similarity as a function of their corresponding frequency and amplitude:

$$\begin{aligned} W_1(X_{lk}, X'_{lk}) &= W_f(X_{lk}, X'_{lk}) \cdot W_a(X_{lk}, X'_{lk}) \\ &= e^{-(\frac{a_{lk}-a'_{lk}}{\sigma_a})^2} \cdot e^{-(\frac{f_{lk}-f'_{lk}}{\sigma_f})^2} \end{aligned}$$

Here we use the Radial Basis Function (Gaussian kernel) $e^{-(\frac{dist(x,x')}{\sigma})^2}$ when converting the distance between two components to its similarity, where is the standard deviation of the component distribution. We then construct a similarity matrix of size $K \cdot L \times K \cdot L$ as our feature space.

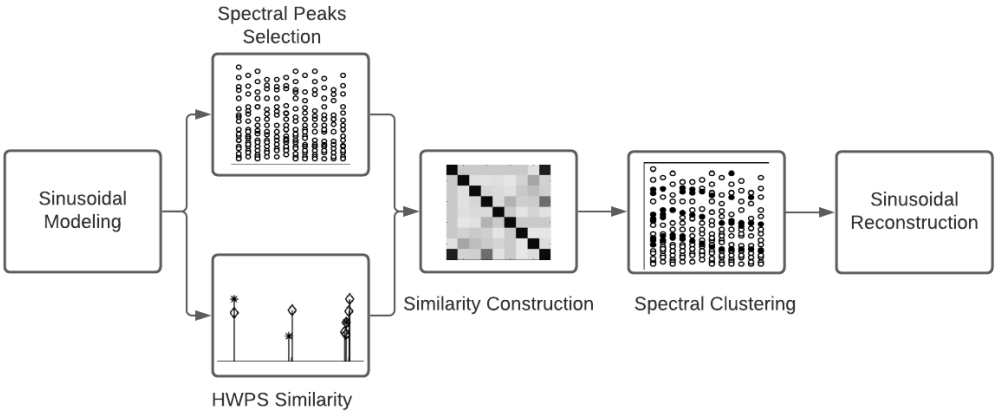

Figure 1. Flowchart of the Sound Source Separation System

2.2.2 Harmonically Wrapped Peak Similarity

To leverage the prior knowledge that music instruments are perfromed harmonically, we apply harmonic cues as a part of the similarity matrix when clustering. According to [6], we incorporated another similarity measure into our clustering mechanism named *Harmonically Wrapped Peak Similarity* (HWPS). The idea behind the HWPS measure is to assign a spectral pattern to each peak based on the frequency context of its corresponding time frame. We then compute a histogram to represent harmonicity of each spectral pattern. The degree of matching between two spectral pattern histograms is used as a similarity measure between the two peaks thus utilizing more spectral information than just the amplitude and frequency of the two peaks. We then use a harmonically wrapped frequency space to align two spectral patterns as they shift when the sound waves change frames. Ultimately, we want the similarity between peaks belonging to the same harmonic complex to be higher than the similarity of peaks belonging to different harmonic complexes.

In order to calculate HWPS, we first compute the shifted spectral pattern $\tilde{F}_{lk}$for each peak X_{lk} based on the set of frequencies $F_{lk} = \{f_{ik}\}$shifted within the frame k:

$$\tilde{F}_{lk} = \{\tilde{f}_{ik} | \tilde{f}_{ik} = f_{ik} - f_{lk} \forall i \in [1, L_k]\}$$

We then wrap the frequency space:

$$\hat{f}_{ik} = mod(\frac{\tilde{f}_{ik}}{h}, 1)$$

where h is the wrapping frequency function and mod is the real modulo function. We also compute the amplitude weighted histogram H_{lk} by discretizing the harmonically wrapped spectral pattern $\hat{F}_{ik}$. The HWPS similarity between the peaks and is defined based on the cosine distance between the two corresponding discretized histograms as follows:

$$W_h(X_{lk}, X'_{lk}) = e^{\frac{c(H_{lk}, H'_{lk})}{\sqrt{c(H_{lk}, H_{lk}) \cdot c(H'_{lk}, H'_{lk})}}}$$

Where $c(H_{lk}, H'_{lk})$ is the dot product between H_{lk} and H'_{lk}. Finally, we incorporate HWPS measure into our similarity matrix:

$$W_2(X_{lk}, X'_{lk}) = W_f(X_{lk}, X'_{lk}) \cdot W_a(X_{lk}, X'_{lk}) \cdot W_h(X_{lk}, X'_{lk})$$

2.3. Reconstruction

After performing spectral clustering and assigning peaks to different groups, we reconstruct each source in the following way. We first mask the STFT matrix according to the selected components, where the components we want to recover is multiplied by 1 (no change) and all other components are set to 0. This is essentially muting the unrelated frequencies. Then we use this masked STFT matrix as input and apply inverse-STFT to reconstruct separate sources.

3. Experimental Results

3.1. Datasets

We create a toy sample of a 4-second piece of audio consisting of a piano, a guitar and a bass. We also used the popular Musdb18 dataset for visualization of actual

3.2. Metrics

To evaluate model performance, we define a loss in the following way. For a test data X_{test}, consisting of n ground truth components $\{X_i\}$ for $i = 1, ...n$, the clustering algorithm will return n separated components $\{X'_i\}$ for

$i = 1, ...n$. Since no label is provided, we need to find a bijective matching between the returned components and the ground truth components. We observe that this is just the stable marriage problem and can be solved in $O(n^2)$ time. Since n is usually small, we just implement a brute force search for experiments. For a given mapping from returned component to ground truth component $X_j = f(X_i')$, the clustering loss can be defined by the cost of matching:

$$L = \sum_{i=1}^{n} ||X_i - f(X_i')||_2^2$$

3.3. Analysis of HWPS

We analyze the effectiveness of Harmonic Wrapped Peak Similarity cue as a similarity metric. We compared the separation loss between constructing the similarity matrix:

1. only using frequency and amplitude[1]
2. only using HWPS
3. using frequency, amplitude and HWPS

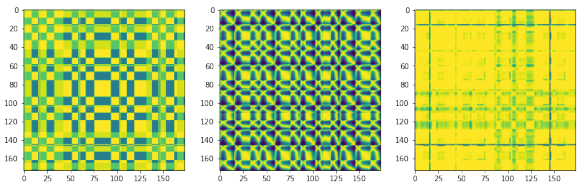

Figure 2. Frequency Similarity (left), Amplitude Similarity (middle), HWPS Similarity (right)

We plot the clustering results in Figure 3-5, where the three axis are amplitude, frequency and time frames. Note that HWPS cannot be plotted because it is only a similarity metric, and to plot its coordinates we would have to do some form of embedding. In Figure 3, we can see that when only amplitude and frequency are considered, the clusters have a clear boundary in this space, which is expected since only only amplitude and frequency distance metrics are used in similarity construction. The boundaries are parallel to the time axis, suggesting that this clustering is stationary with respect to time. It is obvious that this is not an ideal clustering since instruments vary dynamically in music.

In Figure 4, only the HWPS similarity is used for clustering. We can observe that clusters are now "blended in" with each other, indicating that it is possible to group components across frames and frequencies. Recurrent patterns can also be notices along the time axis, indicating some rhythmic structure has been capture. In particular, the red decaying shaped clusters are evidence of the fact that the amplitude of instrument notes often decay with time.

In Figure 5, all similarity cues mentioned above are used for clustering. We can now see while amplitude and frequency differences are captured, the boundaries are not absolute and each component have portions existing across frequency and amplitudes. Intuitively, the blue cluster with relatively high amplitude and low frequency should correspond to the bass, the yellow cluster with high frequency and low amplitude corresponds to the guitar, and the red cluster in the middle represents the piano.

To quantify clustering results, we used the separation loss[2] defined in 3.2 to evaluate the results of the three different similarity constructions. As shown in Table 1, utilizing frequency, amplitude and HWPS achieved the lowest loss. This suggests that HWPS is beneficial for the representation of audio components and is complementary to traditional similarity cues.

Similarity Matrix	Best Match Total MSE
Only Frequency & Amplitude	142.26
Only HWPS	362.96
Frequency & Amplitude + HWPS	134.69

Table 1. Total MSE of different similarity matrices (lower is better)

3.4. Weighed similarity function

Recall in section 2.2.2 we derived a combined similarity function of frequency, amplitude and harmonicity.

While frequency and amplitudes are both distance based metrics and are normalized by their standard deviations, the harmonicity similarity is computed in a different way and there is no reason to believe that having the same weights as frequency and amplitude in the overall similarity function is optimal. We further improve the similarity construction by adding a weight on harmonicity as a hyperparameter α:

$$W_3(X_{lk}, X'_{lk}) = W_f(X_{lk}, X'_{lk}) \cdot W_a(X_{lk}, X'_{lk}) \cdot [W_h(X_{lk}, X'_{lk})]^{\alpha}$$

We tested for different αs and plotted the relationship between the weight of harmonicity and the clustering loss in Figure 6. Results show that $\alpha = 4$ obtains the best separation, indicating that this indeed is a tunable hyperparameter which has influence on the quality of the similarity matrix.

[1] Amplitudes and frequencies are converted to db and bark scale

[2] The distortion loss caused by peak selection can be computed and is constant (=9.459) for every task, so we subtract for every loss below.

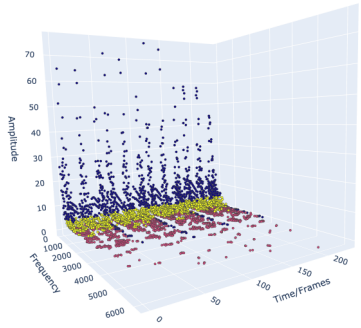

Figure 3. Only using Frequency & Amplitude

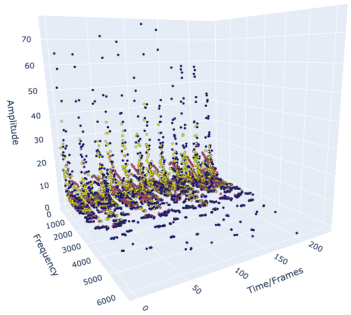

Figure 4. Only using HWPS

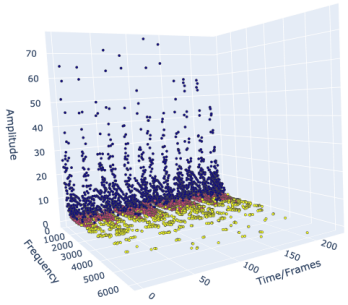

Figure 5. Frequency & Amplitude + HWPS

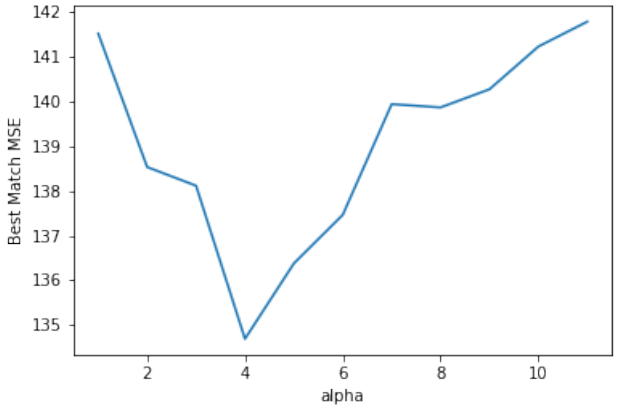

Figure 6. relationship between the weight of harmonicity and the clustering loss

3.5. Clustering Methods

In this section we compare the performance of various clustering methods (all with F&A + HWPS). As shown in Figure 7-9, and results from Table 2, spectral clustering achieved the best results compared to agglomerative clustering and DBSCAN. For agglomerative and DBSCAN, we converted similarity to distance. This result might due to the fact that the low amount of frequency peaks that we selected resulted in limited audio resolution, from which density based clustering methods can not perform as well.

Clustering Method	Best Match Total MSE
Spectral Clustering	134.69
Agglomerative Clustering	139.40
DBSCAN	148.01

Table 2. Total MSE of different clustering methods (lower is better)

3.6. Fast implementation for global clustering

An issue with HWPS is that it is computationally intensive. Computing a pairwise distance between all $L \cdot K$ peaks based on constructed histograms takes $O(L^3K^2)$ time. To cope with large computational complexity, [8] uses texture windows, a window based method that crops the STFT data along the time frames into smaller blocks. Every window is a number of adjacent time frames of STFT coefficients. Then the algorithm computes similarity matrix and perform spectral clustering over the windows respectively. This method speeds up the process but compromises global information by localizing operations. We introduce a fast implementation of the global HWPS similarity, that preserves global information while drastically improving computation time. In the following algorithm we describe our fast implementation.

Algorithm 1: Fast HWPS

Input: F, A `// STFT frequency and amplitude matrix(L, K)`
F_L = stack(F, (0)); `// (L, L, K)`
shift spectrum for all peaks;
F_s = F_L - transpose(F_L, (0,1)); `// (L, L, K)`
compute pairwise minimum;
F_{LK} = reshape(F, (1, $L \cdot K$));
F_{LKLK} = stack(F, (1)); `// (LK, LK)`
h = min(F_{LKLK}, transpose(F_{LKLK}));
compute amplitude weighted histograms;
A_{LKLK} = stack(reshape(A, (1, $L \cdot K$), (1)));
hists = histogram(F_{LKLK}%h, reshape($A_L KLK$));
compute cosine distance;
Sim = sum(hists * transpose(hists));
Output: Sim `// similarity matrix`

Our method benefits from parallelization of matrix operations. However, the apparent disadvantage is the increasing requirement of memory space, making it impossible for

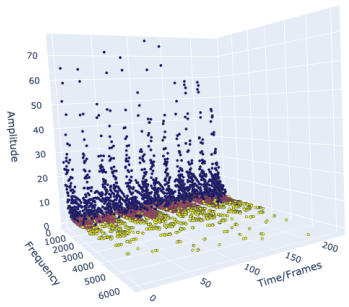

Figure 7. Spectral Clustering

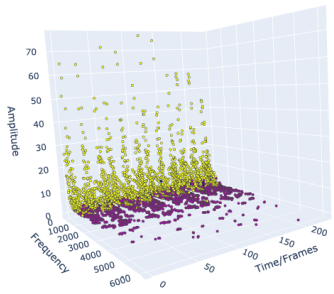

Figure 8. Agglomerative Clustering

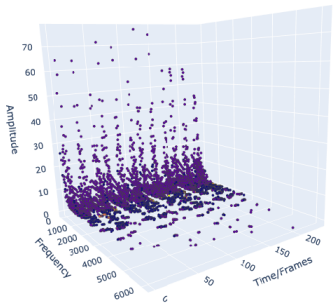

Figure 9. DBSCAN

low-RAM CPUs to execute. To address this, We designed an algorithm to controllably serialize the process by slicing the matrices to trade-off time and space complexities. Note that this is different from [8], since we are still computing the entire similarity matrix instead of a local one. The testing results is shown in Table 2. The re-implementation is based on the authors' official matlab code, where we implemented in python. The Similarity Computational Time is the time the algorithm took to construct HWPS similarity over the entire graph. It can be seen that our fast implementation saves up to 96.8% computational time while slightly improving clustering results. This slight improvement might due to different use of packages and other detailed differences.

Method	Similarity Computational Time	Loss
Re-implementation of [8]	2438.453	139.32
Fast Implementation	79.197	134.69

Table 3. Time Complexity

4. Conclusion

By leveraging harmonic properties of musical sounds, we are able to separate the audio by their instruments. The model we implemented is purely unsupervised and does not involve any form of training. Different from the Blind Source Separation problem, our method does not rely on multiple recordings at all, and only takes in one channel input. We improved [8] by adding a weight to balance harmonicity similarity and other frequency, amplitude. However, the disadvantage of this model is its high time and space complexity, but the fast implementation we introduced is able to control the trade-off by serializing and process sliced segments of similarity matrix. This model can be applied to a wide range of instrument separation scenarios like in karaokes, or use by DJs to create mixtapes. On a high level, this model can potentially be applied to separating sequential data with certain underlying structures.

References

[1] J. S. Calderón-Piedras, D. Orjuela-Cañón, and D. A. Sanabria-Quiroga. Blind source separation from single channel audio recording using ica algorithms. In *2014 XIX Symposium on Image, Signal Processing and Artificial Vision*, pages 1–5, 2014. 2

[2] A. Défossez. Hybrid spectrogram and waveform source separation. In *Proceedings of the ISMIR 2021 Workshop on Music Source Separation*, 2021. 2

[3] A. Défossez, N. Usunier, L. Bottou, and F. Bach. Demucs: Deep extractor for music sources with extra unlabeled data remixed, 2019. 2

[4] R. Hennequin, A. Khlif, F. Voituret, and M. Moussallam. Spleeter: a fast and efficient music source separation tool with pre-trained models. *Journal of Open Source Software*, 5:2154, 06 2020. 2

[5] S. Jain and D. Rai. Blind source separation and ica techniques: a review. *IJEST*, 4, 04 2012. 2

[6] M. Lagrange, L. G. Martins, J. Murdoch, and G. Tzanetakis. Normalized cuts for predominant melodic source separation. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):278–290, 2008. 4

[7] M. Lagrange and G. Tzanetakis. Sound source tracking and formation using normalized cuts. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2007, April 15-20, 2007, Honolulu, Hawaii, USA*, pages 61–64. IEEE, 2007. 3

[8] L. G. Martins, J. J. Burred, G. Tzanetakis, and M. Lagrange. Polyphonic instrument recognition using spectral clustering. In *ISMIR*, 2007. 2, 3, 6, 7

[9] R. McAulay and T. Quatieri. Speech analysis/synthesis based on a sinusoidal representation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(4):744–754, 1986. 2

[10] R. J. McAulay and T. F. Quatieri. Speech analysis/synthesis based on a sinusoidal representation. *IEEE Trans. Acoust. Speech Signal Process.*, 34:744–754, 1986. 3

[11] M. Raad and I. Burnett. Audio coding using sorted sinusoidal parameters. In *ISCAS 2001. The 2001 IEEE International Symposium on Circuits and Systems (Cat. No.01CH37196)*, volume 2, pages 401–404 vol. 2, 2001. 3

[12] Z. Rafii, A. Liutkus, F.-R. Stöter, S. I. Mimilakis, and R. Bittner. The MUSDB18 corpus for music separation, Dec. 2017. 2

[13] D. Stoller, S. Ewert, and S. Dixon. Wave-u-net: A multi-scale neural network for end-to-end audio source separation, 2018. 2

[14] N. Takahashi and Y. Mitsufuji. D3net: Densely connected multidilated densenet for music source separation, 10 2020. 2