Using Dynamic Time Warping to Improve the Classical Music Production Workflow

by

Smriti Pramanick

B.S., Massachusetts Institute of Technology (2018)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2019

Author .....
Department of Electrical Engineering and Computer Science
August 9, 2019

Certified by .....
Eran Egozy
Professor of the Practice, Music Technology
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Using Dynamic Time Warping to Improve the Classical Music Production Workflow

by

Smriti Pramanick

Submitted to the Department of Electrical Engineering and Computer Science
on August 9, 2019, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The current music production workflow, comprising recording, editing, mixing, and mastering music, requires a great deal of manual work for the sound engineer. This thesis aims to bring some recent advances in Music Information Retrieval (MIR) techniques to music production tools, with the goal of streamlining the current process followed by sound engineers. We explored all areas in the music production workflow (with a focus on classical music) that could benefit from digital signal processing (DSP) and MIR-based tools, built and iterated on these tools, and transformed the tools into products that are beneficial and easy to use.

We collaborated with the Boston Symphony Orchestra (BSO) sound engineers to gather requirements for this work, which led to the identification of our two tools: an automatic marking transfer (AMT) system and an audio search (AS) system. We then collaborated with other potential users for both AMT and AS tools, including sound engineers from radio stations in the Boston area. This enabled us to identify additional workflows and finalize requirements for these tools. Based on these, we created successful standalone applications for AMT and AS.

Thesis Supervisor: Eran Egozy
Title: Professor of the Practice, Music Technology

Acknowledgments

I would like to thank my thesis advisor, Eran Egozy, for his invaluable guidance and support, and the many sound engineers in the Boston area that I had the privilege of working with. They selflessly gave their time and provided tremendous help and feedback in shaping this thesis work and the tools we created.

I would also like to thank my parents, without whose encouragement I would not be here today: they helped me realize my passions for music and computer science, which combined, became the biggest motivation for this work.

Contents

List of Figures

Chapter 1

Introduction

The current music production workflow, comprising recording, editing, mixing, and mastering music, requires a great deal of manual work for the sound engineer. This thesis aims to bring some recent advances in Music Information Retrieval (MIR) techniques to music production tools in order to streamline this process. Our goal is to explore all areas in the music production workflow (with a focus on classical music) that could benefit from digital signal processing (DSP) and MIR-based tools, to build and iterate on these tools, and to transform the tools into products that are beneficial and easy to use.

This chapter outlines the various steps of the music production process, followed by a discussion of the current tools used by sound engineers and one of the biggest pain points of the process. It concludes with a section on why this thesis work focuses on the music production workflow for classical music in particular.

1.1 The Music Production Process

The process for producing a piece of music is to record the piece, edit separate tracks, mix the tracks together, and finally master the mix [30].

1.1.1 Step 1: Record the piece

In the past, full pieces were recorded with one or two microphones and in one to two takes. However, nowadays it is more common to record with several microphones to create a multitrack recording, with a track per microphone, and to record several times.

Recording with multiple tracks allows each instrument to be recorded separately. For example, an engineer could record each instrument of a string quartet separately and then combine them into one recording later. In addition, multitrack recording allows engineers to later make changes to each track independently. With each instrument in its own track, an engineer can isolate an instrument and change just that instrument's recording.

The goal of recording the same section of music multiple times (each time is referred to as a "take") is to capture the musician's best possible rendition of the piece. Since each instrument is recorded in a separate track, this allows the engineer to select the best of each instrument for the final product.

Current technology also allows for recordings to be just parts of a piece: an engineer no longer has to record full run throughs (from the beginning to the end of a piece) with all the musicians. Instead, they can record parts of a piece and stitch them together later.

1.1.2 Step 2: Edit the tracks

In this phase, sound engineers touch up the recorded tracks. This involves tasks like correcting out-of-tune notes, removing extra noise (e.g., chairs creaking), removing entire tracks (e.g., a track with mistakes or out-of-tune instruments), etc. This is the step in which an engineer isolates tracks and changes them individually, perhaps splicing together multiple takes of the same music to create the best overall take, or removing mistakes by substituting portions from different takes.

1.1.3 Step 3: Mix the tracks to create one recording

Sound engineers then mix separate tracks together to create one, cohesive recording of the piece. First, they must choose the best takes for each section of music (and sometimes for each instrument). Then the rest of the mixing process involves tasks like balancing the tracks so that the tracks are at appropriate volume levels with respect to each other, compressing tracks to normalize the volume range or perceived loudness of each instrument, etc.

1.1.4 Step 4: Master the mix

After all the tracks have been mixed together, the tracks have to be rendered into a single stereo file and converted to the appropriate sample rate/bit depth (e.g., 44.1 kHz/16 bits for a CD). This process is called mastering, during which time the engineer might also add steps like adjusting the overall dynamics of the entire stereo mix, or adding overall frequency adjustment effects like "equalization", etc.

1.2 Tools for Producing Music

This section describes the current technologies that sound engineers use to produce music.

1.2.1 Digital Audio Workstations

Sound engineers use a digital audio workstation (DAW) to record, edit, mix, and master a piece [34]. Some examples of DAWs are Pro Tools and Pyramix [5, 33]. These DAWs provide an interface for multitrack recording, editing each track, and stitching together tracks. The user interface for Pyramix is shown in Figure 1-1 below, along with hardware units used for recording, effects, etc.

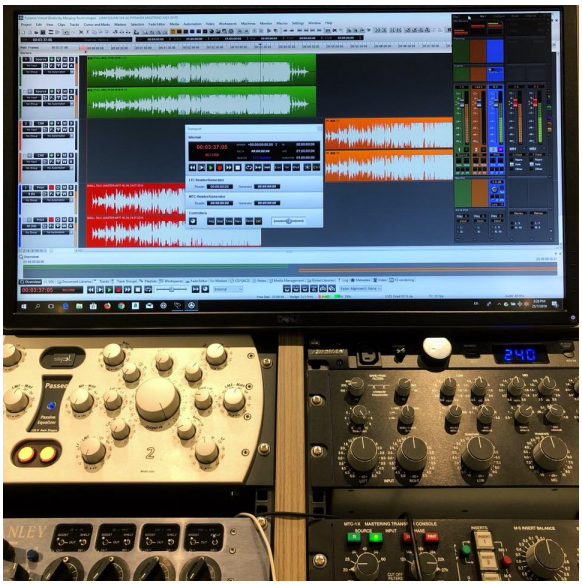

Figure 1-1: Pyramix interface for music production (top half), and hardware units used for recording, effects, etc. (bottom half) [33].

1.2.2 Virtual Studio Technology Plugins

Sound engineers also use Virtual Studio Technology (VST) plugins, which integrate software synthesizers and effects into a DAW [35]. VST plugins are supported by most DAWs [35], so a developer can build one plugin to be used across multiple DAWs. These plugins provide additional features and functionality to use in the music production process, such as mix monitoring tools like ISOL8 [28], which is shown in Figure 1-2 below.

Figure 1-2: ISOL8, a VST plugin by TBProAudio for advanced mix monitoring [28].

There is a rich market of third party developers who make specialized VST plugins like ISOL8. VST plugins fall under three main categories [35]:

- VST instruments, which generate audio (e.g., virtual synthesizer),
- VST effects, which process audio (e.g., reverbs and phasers; ISOL8 falls under this category), and
- VST MIDI [1] effects, which process MIDI messages.

1.3 Streamlining the Music Production Process

One of the biggest pain points for sound engineers during the music production process is identifying the content of all the tracks they have while mixing the tracks together [27]. Currently, there is no universal solution for this problem: none of the available VST plugins provide an automatic way to detect which track aligns with which part of the music in another track, and only one DAW (Sequoia) has piloted such an audio alignment system [11]. Thus, for the large majority of use cases, sound engineers have to devise a manual marking scheme while recording, to aid them in the mixing process later. These current marking schemes require a lot of manual work: the score and/or tracks must be marked by hand during the live recording session, and later interpreted in the mixing stage. These approaches are described in more detail in Sections 2.1.1 and 2.1.2 of Chapter 2.

Using DSP algorithms and MIR techniques, we can reduce this manual work. In this thesis, we describe the use of Dynamic Time Warping (DTW) to help identify relative content of several tracks [13]. If an engineer is working with a section of a piece and wants to find all matching instances of that section in other tracks, they typically have to search through all the tracks manually, looking for the same waveform or searching through annotations they made on the score or tracks before

[1] MIDI stands for Musical Instrument Digital Interface, which is a way to connect devices that make and control sound, so that they can communicate with each other [38].

the editing and mixing steps. We propose using DTW to solve this audio matching [2] problem and find the matches automatically, saving time and effort for the engineers.

DTW can also be used to synchronize two recordings of a piece [13]. "Synchronization" in this context is the process of building a mapping of sequence points in one recording to those in another: for a given sequence point in one recording, this mapping will give the musically corresponding sequence point in the other [3]. For example, we may have two recordings of a piece: one orchestral, and one solo piano. Not only may these recordings differ in timbre and dynamics, but they may also differ in tempo, so that the same musical moments may be in different temporal locations; in such a case, we can use DTW to synchronize the two pieces. Doing this will tell us, for example, where measure ten appears in both pieces. Thus, if a sound engineer has manually added markings (timed metadata) to one recording of a piece, we could use DTW to transfer these markings to the other recordings of the same piece by synchronizing the two pieces. This means that the sound engineer will only have to mark one recording of a piece during a recording session and can later use DTW to transfer these markings to other recordings of the same piece.

We discuss DTW and explain the algorithm in more detail in Section 2.2 of Chapter 2.

1.4 Why Classical Music?

The tools developed as part of this thesis are most relevant to classical music production. This is because other genres like popular music follow a different workflow. The pop music production workflow often involves creating a mix track by track (i.e., working with each instrument/vocals separately) and working without a formal, written out score [27]. Many pop music tracks might be first created in the DAW itself. For example, a drum track might be created and then all performers would listen to

[2] Audio matching is the task, given a query (an audio snippet), of finding all other snippets that musically correspond to that query [13].

[3] The corresponding sequence point will be the same point musically, if not the same point in time.

that drum track through headphones while playing or singing additional tracks on top of that base track. In such a workflow, the need to synchronize different tracks is usually not an issue. Pop music pieces also tend to be smaller and simpler, so the process is faster and may not need the aid of such tools [27].

Another reason why we chose to focus on classical music is because DTW (the underlying algorithm that allows us to create such tools) is at its core a music synchronization algorithm. Thus, it assumes two recordings of the piece will be musically similar, only differing in aspects like tempo, timbre, and dynamics [13]. Therefore, a genre like jazz that incorporates a lot of improvisation may not lend itself to such an algorithm.

Chapter 2

Related Work

This chapter discusses the current tools for sound engineers producing classical music, and introduces the Dynamic Time Warping (DTW) algorithm that may be used to create more tools.

2.1 Current Tools

2.1.1 Handwritten Notes and Visual Cues

The standard for creating markings during a live recording is to take notes by hand. This means that during a recording session, the sound engineer or assistant who has a copy of the musical score will mark the score with "+"s and "−"s along with a track number, to indicate if the track was a good or bad run of the section (see Figure 2-1 below for an example). This helps them ensure they have at least one good run of every section. In addition, later in the mixing process, they can use this metadata to determine which tracks to look at and ultimately choose in the final mix.

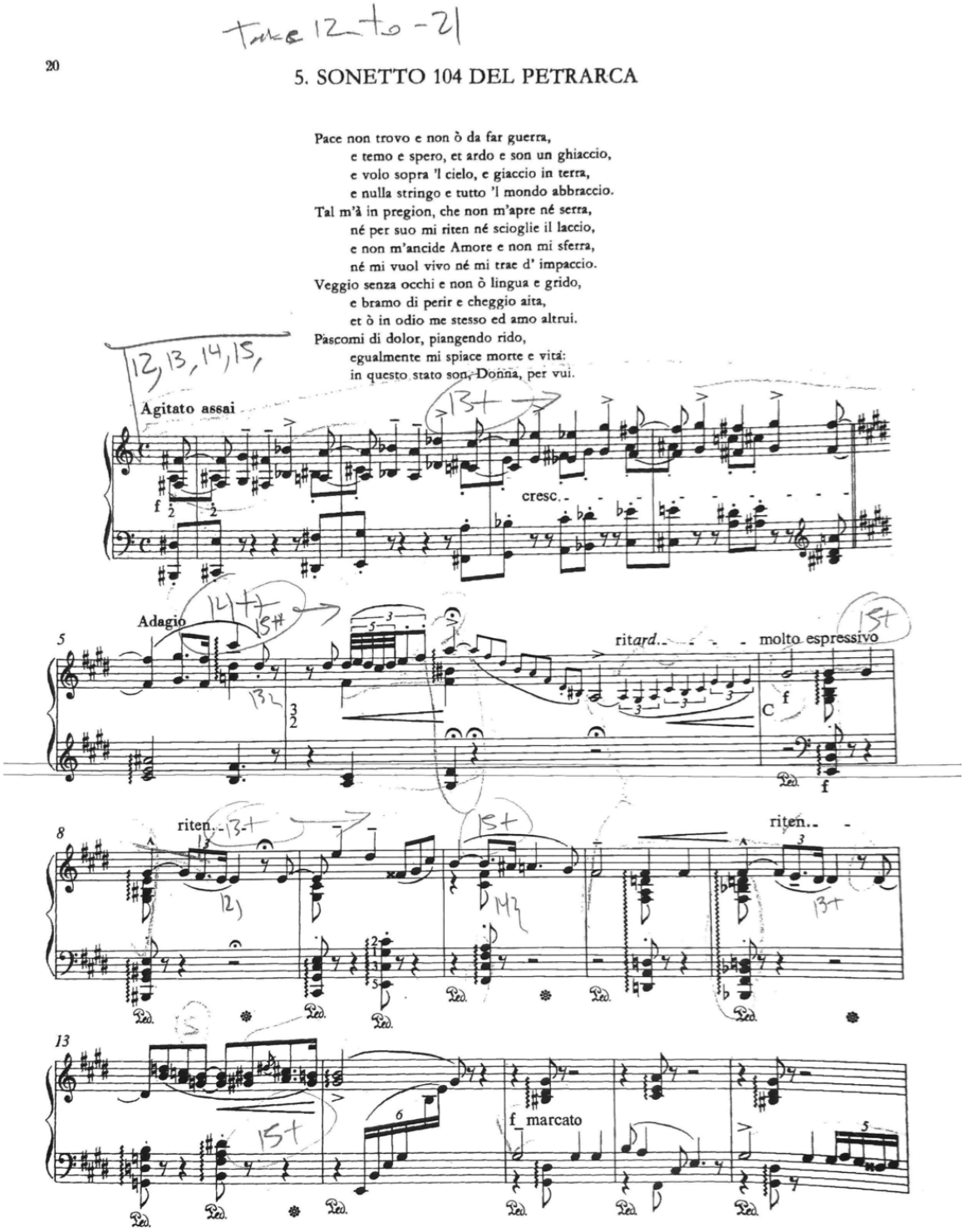

Figure 2-1: Annotated score: marked with "+"s and "−"s along with a track number, to indicate if the track was a good or bad run of the section. Photo courtesy of Antonio Oliart [14].

As DAWs typically show the waveform of the loaded audio, sound engineers also start to learn the waveform of a piece in order to quickly identify and match different tracks (i.e., visually recognize and differentiate between sections of a piece). This means that during the editing and mixing steps, if the engineer wants to find the best run and recording of a section, they must either look at the waveform and search for the same snippet visually, or read through their annotated scores and logs to find all occurrences of that section.

2.1.2 Digital Markings

An addition to the above marking scheme is to also add markings directly to the audio recording (i.e., directly in the DAW software as opposed to by hand), indicating which timestamps correspond to which rehearsal numbers/letters [1]. This way, during the mixing process, the engineers can more easily identify parts of a song via these rehearsal number/letter markings embedded in the tracks (they are shown visually in the DAW; see Figures 2-2 and 2-3 below), instead of having to study, memorize, and subsequently recognize the waveforms.

Digitizing the marking system by using existing DAWs to annotate recordings directly in this way reduces the work a bit [27]. However, engineers still have to do this manually for each recording of the piece (in full or part) during the recording session. As far as we know, no tools have been created to help automate this process and reduce the amount of manual work.

[1] Rehearsal numbers/letters are provided in a musical score for classical music, typically at the beginning of a measure.

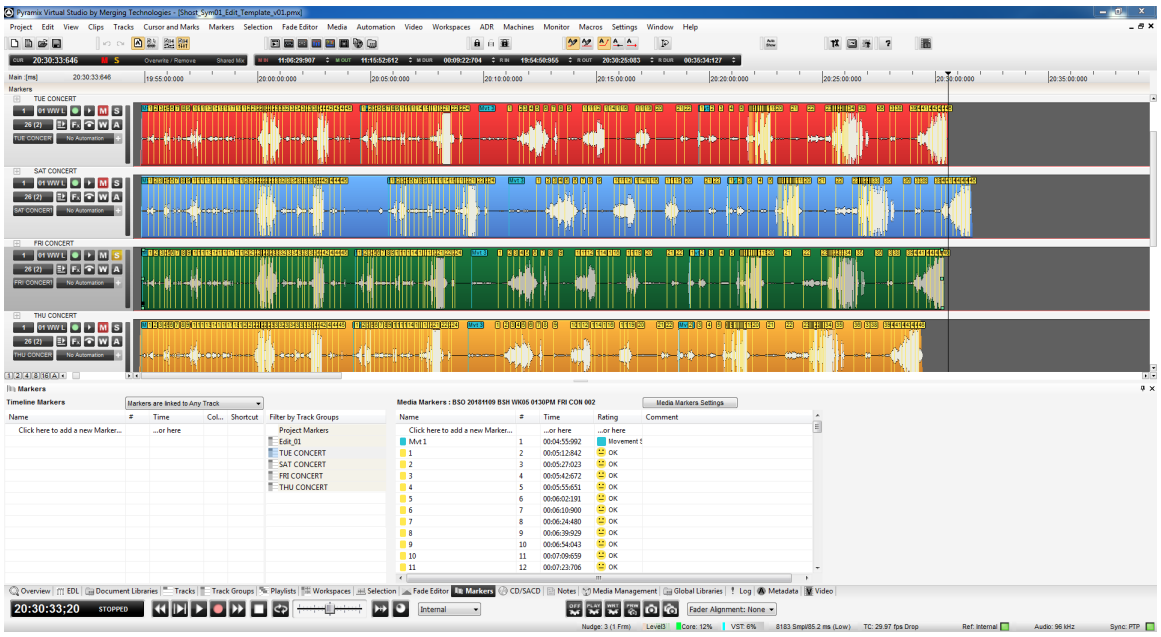

Figure 2-2: Annotated tracks in Pyramix. In this example, each track is a different recording of the same piece, and each track is annotated by hand after recording (i.e., each rehearsal number marking is created for each track). Screenshot courtesy of the BSO [27].

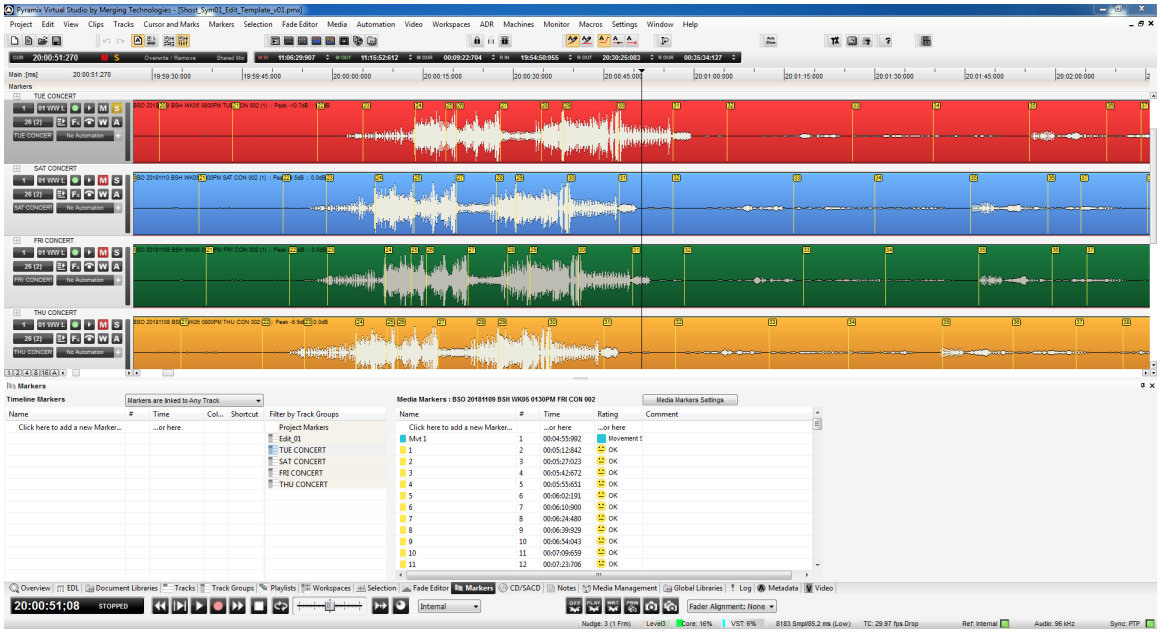

Figure 2-3: Annotated tracks in Pyramix (zoomed in view of Figure 2-2). Screenshot courtesy of the BSO [27].

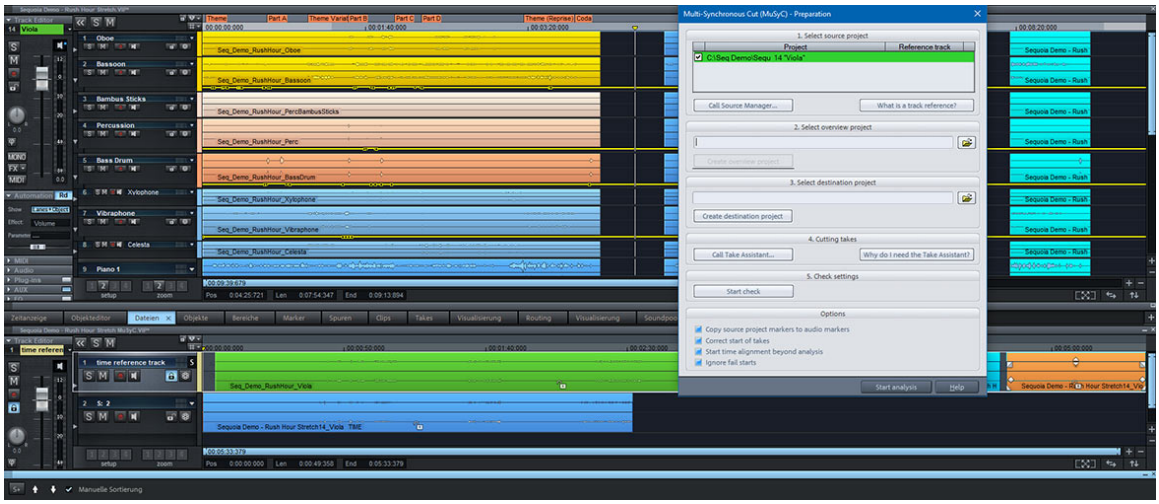

Figure 2-4: MuSyc feature in Sequoia [11].

2.1.3 Sequoia's Multi-Synchronous Cut

The Sequoia DAW has a feature called Multi-Synchronous Cut, or MuSyc (shown in Figure 2-4 above), targeted for classical music production. This feature can be used to find similar audio in a project. MAGIX (the company that makes Sequoia) has not released the underlying algorithms used to do this, but the feature description on their website states that it analyzes source audio material and then searches through the user's project for musically similar material. It then organizes the matches [2] by similarity to the query [11], as shown in Figure 2-5 below.

[2]If the user's project has one take for all recordings, the user can use the "Take Assistant" to divide it into smaller takes [10].

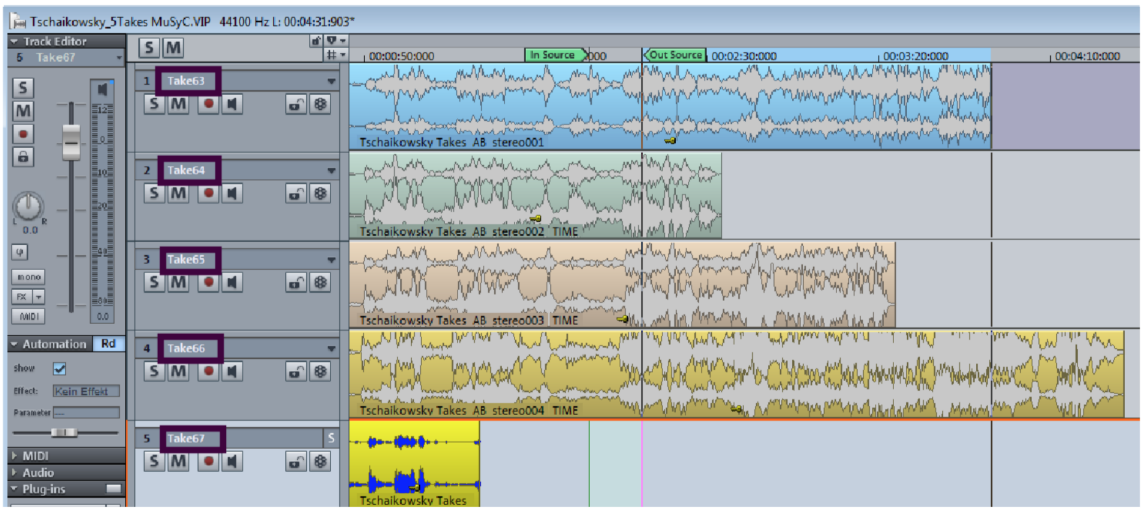

Figure 2-5: Matching audio (takes with the same musical content) are stacked on top of each other in the overview project [10].

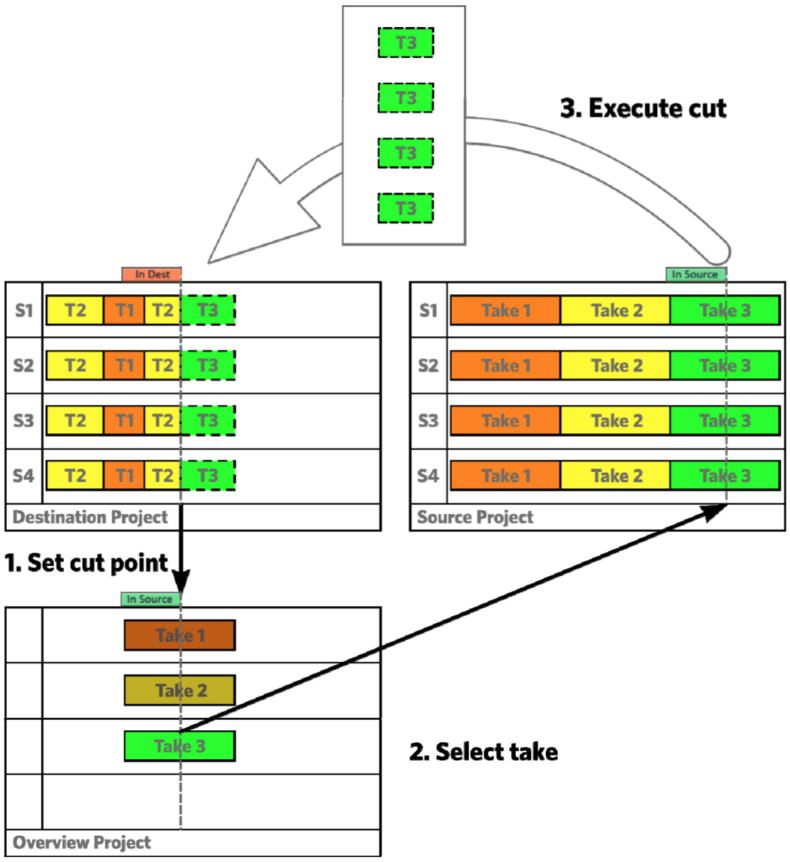

Figure 2-6: The flow of the MuSyc feature [10].

The flow to using MuSyc is as follows (diagrammed in Figure 2-6 above):

- Identify the problem spot in the mix to replace and set the cut point at that spot in the destination project [3].
- Use MuSyc to create an "overview project", where similar audio for the musical section is displayed.
- Choose a take for this section.

Once the user has chosen which audio take to use in the overview project, it will be automatically synchronized to the source project, and once the cut is executed, it

[3]Sequoia is a source-destination DAW, where there is a source project (where all the recordings live) and a destination project (where the final mix is created).

will be inserted into the destination project [10]. In this way, engineers can quickly explore and choose from all recordings of a musical section.

We investigated MuSyc in more depth via conducting an interview with a Sequoia user, Patrick Keating. In particular, we wanted to learn more about the experience using the feature. A more detailed recap of this interview is described in Section 3.2.2 of Chapter 3.

2.2 Dynamic Time Warping

DTW is a robust algorithm that can be used for synchronizing two recordings of a piece of music [13]. As discussed in Section 1.3 of Chapter 1, synchronization is building a mapping of sequence points in one recording to another: for a given point in one recording, this mapping will give the musically corresponding point in the other [4].

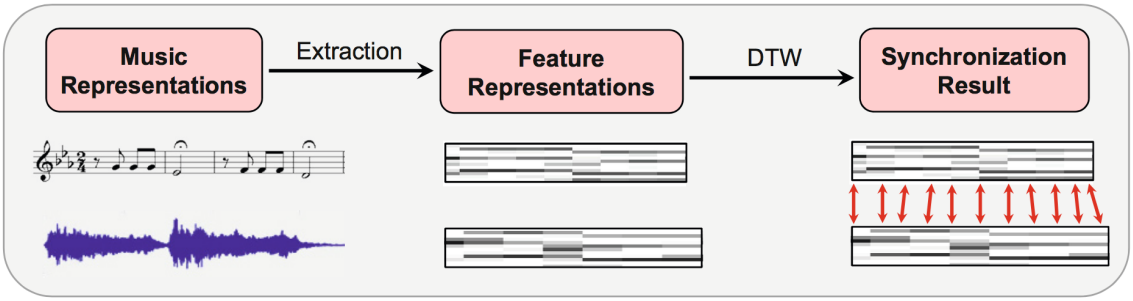

Figure 2-7: Audio Synchronization Pipeline [13]. This process converts music of potentially different representation to the same feature representation (here, chromagrams), and then runs the DTW algorithm to identify points of correspondence from one recording to another. Figure used by permission of Meinard Müller [13].

The synchronization process involves converting the two recordings to the same feature representation (usually at a sampling rate lower than the audio sampling rate; e.g., chromagrams [5]) to allow for comparison and then running the DTW algorithm,

[4]The corresponding point will be the same point musically, if not the same point in time.

[5]A chromagram is a representation of audio which distributes frequencies into twelve pitch bands over time. It is a tool used to analyze music: it captures the melodic and harmonic components of music and is typically invariant to changes in timbre, instrumentation, and dynamics [36].

as shown in Figure 2-7 above. We provide an overview of this process below, and describe the technical details of the implementation in Python in Chapter 4.

The goal of DTW is to find an alignment between the two sequences, X and Y. This alignment will be a sequence of pairs of elements in X and Y. Since these sequences may be of different lengths, the alignment may be nonlinear (skipping or repeating elements), as shown in Figure 2-8 below. Our sequences are chromagrams, but DTW can be used with any kind of time series [6].

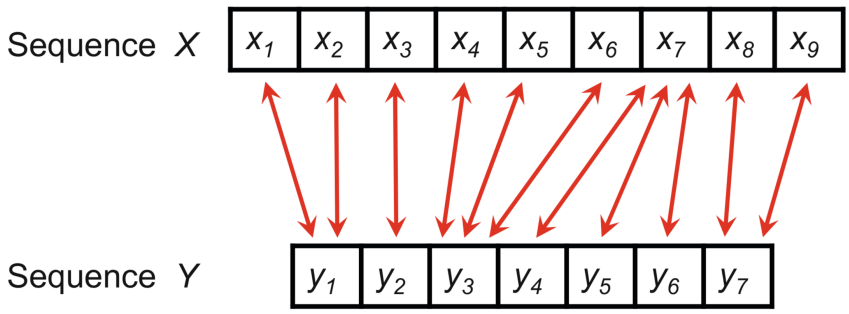

Figure 2-8: The alignment of two sequences, X and Y. Figure used by permission of Meinard Müller [13].

To determine the alignment between elements of the sequences, we need a cost or similarity metric to compare the elements with each other. Elements that are more similar to each other should have a lower cost than elements that are less similar. In this audio synchronization application of DTW, the similarity metric we use is the cosine distance [7]:

$$1 - \frac{x \cdot y}{||x|| \cdot ||y||}$$

We can calculate this cost metric for each pair of elements from the sequences. If sequence X is N elements long, and sequence Y is M elements long, this will yield a matrix of dimensions $N \times M$, which is known as the cost matrix. As shown

[6] In fact, DTW is used outside of the audio synchronization realm, in automatic speech recognition, data mining, information retrieval, bioinformatics, and several other fields [13].

[7] Since the cosine distance measures the angle between two vectors and disregards the length of the vectors, using it with chromagram features gives the difference in energy distributions, instead of in local energy, which is a valid metric in comparing sound [13].

in Figure 2-9 below, since the two recordings are of the same piece, they have a similar progression apart from a global difference in tempo, so there are lower costs around the diagonal of the cost matrix (and the alignment will not stray far from the diagonal).

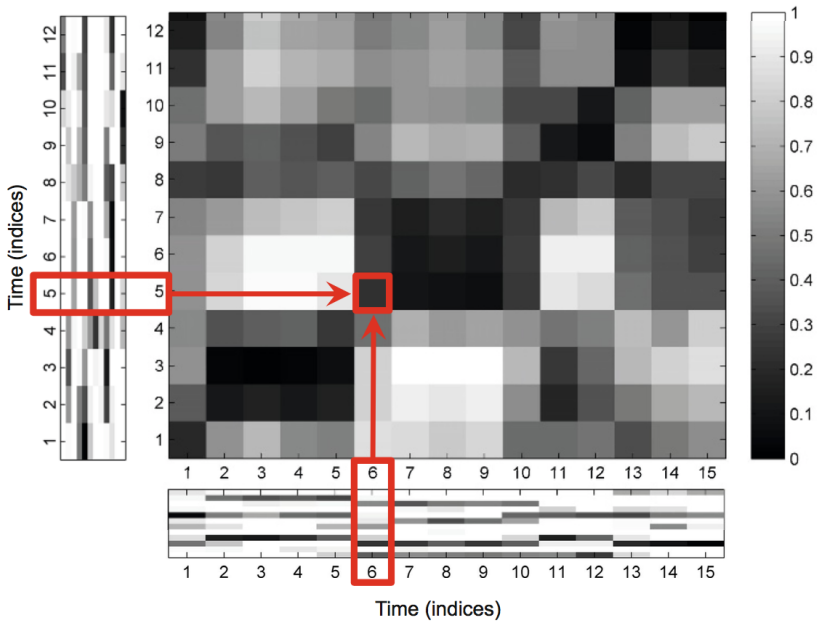

Figure 2-9: DTW cost matrix between two chromagrams (shown along the axes) [13]. The cost between index 5 of the y-axis and index 6 of the x-axis is shown in the red box. Figure used by permission of Meinard Müller [13].

Given this cost matrix, we can synchronize sequences X and Y by finding the alignment with the minimal overall cost (since the more similar two elements are, the less their cost will be). Intuitively, viewing the 2D matrix as a "height map", where height corresponds to cost, this alignment is the valley of the cost matrix (see Figure 2-10 below).

Valid alignments are ones which abide by the following:

- The first elements of X and Y and the last elements of X and Y must align with each other.
- The alignment cannot go "backwards in time": starting from $(1, 1)$, the alignment path can only go up or to the right (i.e., the next point in the path

$(1 + i, 1 + j)$ is only valid if $i, j \geq 0$).

- Each element of X is aligned with at least one element of Y, and vice versa.

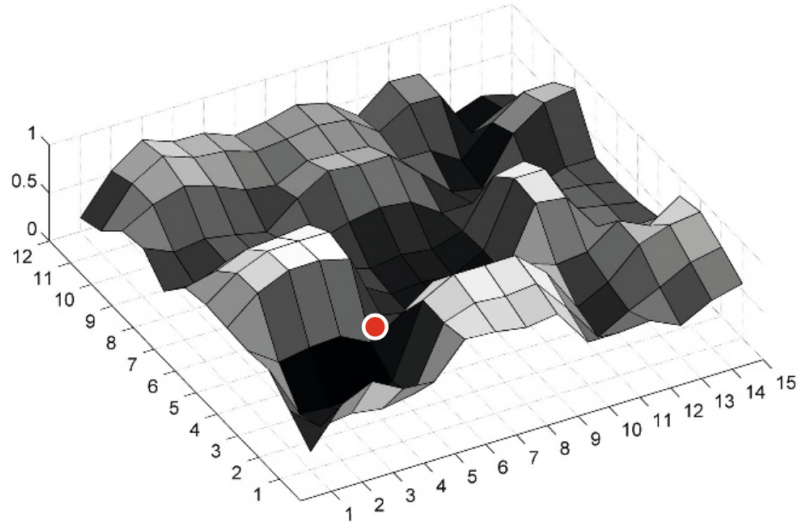

Figure 2-10: DTW cost matrix viewed as a "height map", where height corresponds to cost [13]. The alignment with the minimal overall cost lies in the valley of the matrix. Figure used by permission of Meinard Müller [13].

There are several potential valid alignments (paths of this cost matrix). The optimal one will be the path with the total lowest accumulated cost (there may be more than one). To find this optimal path, we could calculate all paths and accumulated costs, but that approach would be exponential in the lengths of sequences X and Y. Alternatively, we can use dynamic programming to create a quadratic-time algorithm for this optimal path problem (i.e., $O(NM)$ given sequence X is of length N, and sequence Y is of length M). This will use dynamic programming by building the optimal path upon optimal subpaths. We introduce an accumulated cost matrix D, where each cell of the matrix represents the accumulated cost associated with the optimal path ending in that cell (i.e., any cell $D(n, m)$ represents the accumulated cost of the optimal path from $D(1, 1)$ to $D(n, m)$). We calculate D from the cost matrix C iteratively, according to the following three equations (given sequence X is

of length N, and sequence Y is of length M):

$$D(n,1) = \sum_{k=1}^{n} C(k,1) \text{ for } n\epsilon[1:N]$$

$$D(1,m) = \sum_{k=1}^{m} C(1,k) \text{ for } m\epsilon[1:M]$$

$$D(n,m) = C(n,m) + min \begin{cases} D(n-1,m-1) \\ D(n-1,m) \\ D(n,m-1) \end{cases}$$

The first two equations initialize the left-most and bottom-most rows of D, respectively. The third equation calculates the accumulated cost of the remaining cells $D(n,m)$ from $C(n,m)$ and the least of $D(n-1,m-1)$, $D(n-1,m-1)$, and $D(n-1,m-1)$, which are the accumulated costs of the optimal alignment paths leading up to $D(n,m)$ (the paths from the diagonal, from below, and from the left, respectively). We keep track of which option we choose from these three in another backtracking matrix B while computing the accumulated cost matrix D. After we have built up D (and in turn B), we can use B to backtrack from the last point of the alignment $D(N,M)$ to the first point $D(1,1)$ to find the optimal path from $D(1,1)$ to $D(N,M)$. This optimal path gives the points of correspondence between the two sequences, thus aligning them in time.

Figure 2-11 below shows the cost matrix, accumulated cost matrix, and final alignment of two series, as an example.

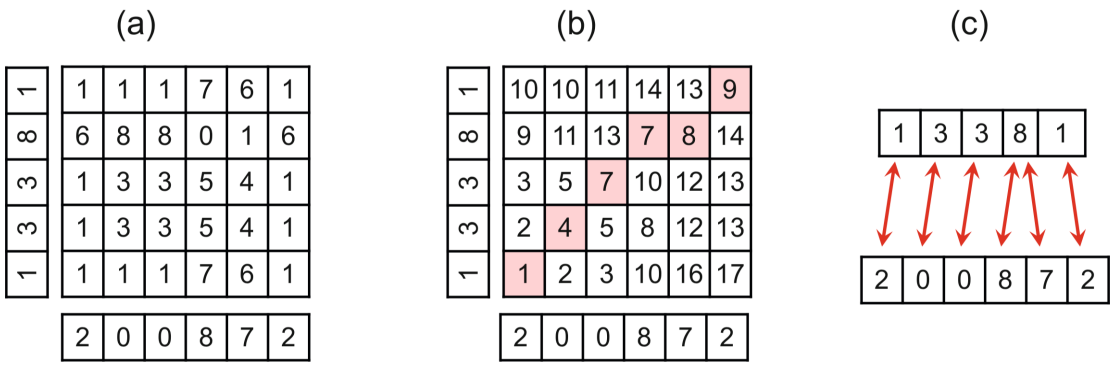

Figure 2-11: (a) Cost matrix between two series, (b) Accumulated cost matrix between two series, with the optimal alignment highlighted in red, and (c) The final alignment of the two series [13]. Figure used by permission of Meinard Müller [13].

2.2.1 Applications of DTW

As described above, we can use DTW to synchronize two recordings of a piece. The resulting warping path creates a correspondence between the two recordings, and if one recording is digitally marked, we can use this correspondence to "translate" the time of a particular musical moment in that recording to the time for the same musical moment in the other recording. This will eliminate the need for engineers to manually mark all recordings of the same piece.

Note that in this application of DTW, we are finding a global alignment of two full recordings (e.g. a full recording X to another full recording Y, as shown in Figure 2-12 below). We can also use DTW to find the alignment of an audio snippet to a part of a full recording [13] (e.g. a snippet X to a part of the recording Y [8], as shown in Figure 2-13 below). This is done by running the DTW algorithm several times, with the snippet and each of the recordings of interest, to find the best match(es). This can be used for audio matching, which will help engineers in finding similar audio.

Thus, these are two ways in which DTW can help streamline the classical music production process, and these led to the creation of the two tools in this thesis: Automatic Marking Transfer (AMT) and Audio Search (AS), which are introduced in Chapter 3.

[8]The part of the recording could be the full recording itself, if X matches with exactly all of Y.

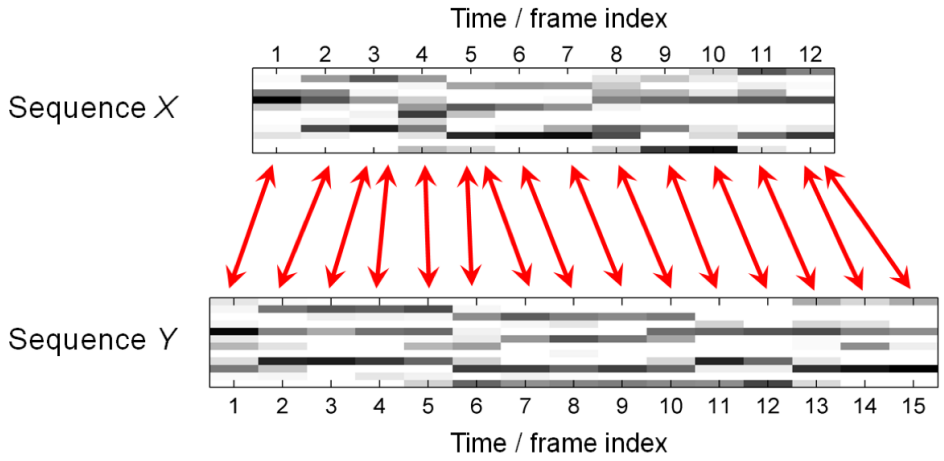

Figure 2-12: Audio Synchronization via DTW. This figure shows the alignment of Sequence X to Sequence Y [13]. Figure used by permission of Meinard Müller [13].

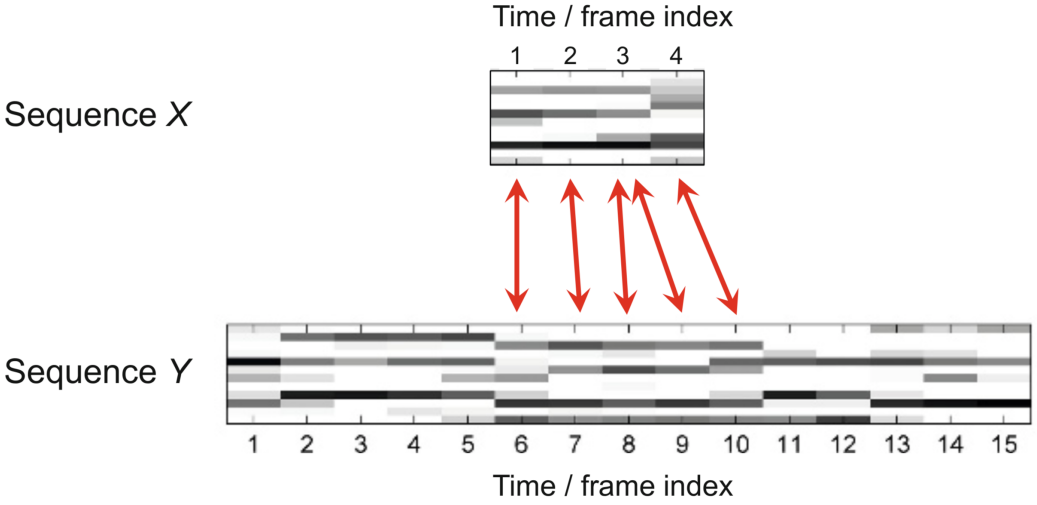

Figure 2-13: Audio Matching via DTW. This figure shows the alignment of Sequence X to a subsequence of Sequence Y [13]. Figure used by permission of Meinard Müller [13].

Chapter 3

Technical Approach

We collaborated with the Boston Symphony Orchestra (BSO) sound engineers to understand their workflow, which led to the identification of two potential tools for improving the classical music production workflow: an automatic marking transfer (AMT) system and an audio search (AS) system. AMT automatically transfers user-created markings (timed metadata) from one recording of a piece to other recordings of the same piece (which may vary in tempo, timbre, and dynamics). AS searches a project for all occurrences of a musical section of a piece that match an audio query snippet (i.e., audio matching).

We then collaborated with other potential users of both AMT and AS tools, including sound engineers from radio stations in the Boston area. This enabled us to identify additional workflows and finalize requirements for these two tools. Based on these, we created standalone applications for AMT and AS.

This chapter details our approach, from the beginnings with the BSO to generalizing to other workflows and formulating our final vision for AMT and AS.

3.1 Beginnings with the BSO

Initial requirements for AMT and AS were obtained by collaborating with the BSO engineers, who use the Pyramix DAW.

We first met with BSO's Chief Recording Engineer, Nick Squire, in November 2018

[27]. The goal of this meeting was to observe and hear about Mr. Squire's workflow in order to discover the places where we could use MIR and DSP techniques to help. We also wanted to learn more about the community of sound engineers in order to understand the audience for such audio engineering tools. Finally, we hoped to form an initial vision of the tools and how they may be incorporated into an engineer's workflow.

The BSO records concerts for three primary purposes: archives, weekly live radio broadcasts, and commercial releases. We focused on the workflow for archives and commercial releases, since these require a lot of work and post-processing after the recording step. For this workflow, during a live performance, Mr. Squire and the BSO's Associate Audio Engineer, Joel Watts, record multitrack (60 tracks) audio through Pyramix, and mix the live mix down to a stereo mix. After the performance, they work on editing and remixing.

For each piece the BSO performs, they record 4-5 performances (every rehearsal and formal concert in which the piece is played). The final mix is created from the best parts of these 4-5 recordings. During the performances, in order to aid with editing and mixing later, Mr. Squire and Mr. Watts create markings in Pyramix for each rehearsal number (the technique described in Section 2.1.2 of Chapter 2). Pyramix provides a way to add markings to indicate good/bad runs of a section, but the BSO engineers have "hacked" this system to use it to indicate rehearsal numbers instead, as shown in Figures 2-2 and 2-3 in Chapter 2.

Mr. Squire and Mr. Watts cannot always get all the markings inputted during the live performances (working at the pace of the piece without interruption can be difficult). Hence, before editing and remixing, they go through each recording and manually add and correct the markings as needed. This is very time consuming, taking up to 4 or 5 hours for each recording [27]. In describing this process to us, Mr. Squire mentioned that it would be wonderful to be able to just mark one of the recordings and then transfer those markings to the other recordings.

Mr. Squire also described another pain point: matching smaller takes of audio to a larger piece of music. Often, during rehearsals, they will record just parts of

the piece, and later while editing/mixing, it can be hard to determine what part of a piece those snippets align with. He stated that the ability to match an audio snippet to a larger piece would be a significant help.

To address these two issues, we explored two systems: one to transfer markings and one to search for an audio snippet in other (primarily larger) sections of audio. We formalized these into the AMT and AS systems. Mr. Squire believes that such tools would be extremely helpful; in fact, he has been thinking about the possibility of such tools for the past ten years. He also believed they would be helpful not only to him and Mr. Watts, but also to the larger classical music production community. There are about fifty thousand such engineers in the US [27].

We also wanted to form an idea of what these tools would look like, specifically, how they would fit into the engineer's workflow. The primary DAW for classical music production is Pyramix [27]. This is because it works well for long pieces and for reading off score during the production process (and almost all classical music has an associated score). However, there are an array of DAWs used in addition to Pyramix: Pro Tools, Nuendo, Sequoia, and more, so it was important that the tools we create be DAW-agnostic. We considered integrating with existing DAWs via technologies such as VST plugins (see Section 1.2.2 of Chapter 1) or Steinberg's SKI SDK [29], or creating our own standalone applications (with our own GUI). Before deciding between these two routes, we decided to prototype the tools as Python scripts (implementation detailed in Chapter 4). In parallel, we carried out additional interviews with other engineers (potential users) to help us decide which path to take.

3.2 Generalizing to Other Workflows

After meeting with the BSO engineers and while prototyping AMT and AS, we carried out additional interviews with other engineers (potential users) to generalize AMT and AS to other workflows and DAWs. In particular, we were most interested in how the tools could be packaged to fit seamlessly into the workflows of all sound engineers. The tools needed to be easy to use and universal (i.e., should work with any DAW).

Section 3.2.1 discusses the different packaging options we considered and Section 3.2.2 gives a more detailed account of the additional interviews, which were key in helping us make our final decision.

3.2.1 Options for Packaging the Tools

The ideal packaging option would be to wrap AMT and AS into the already universal VST plugins. This way, AMT and AS could become a part of every DAW and easily integrate into a sound engineer's workflow. The purpose of VST plugins is to change audio in some way, so using VST for AMT and AS was not the traditional use case of the plugin (neither of these processes changes audio). However, since VST plugins are the primary way for a third party developer to integrate with DAWs, we wanted to keep this option on the table.

Unfortunately, after further research and conducting interviews, we discovered that VST plugins may not provide all the capabilities we need. For AMT, we needed the ability to create markings. Markings are actually specific to each DAW, so in building a VST plugin for AMT, there would have to be different versions for each DAW. For example, markings can be used via associated MIDI files in Pro Tools, but the best way to interface with Pyramix is Pyramix's own proprietary markings called "MediaMarker" [27]. Users of Steinberg DAWs (e.g. Nuendo or Cubase) can use the SKI SDK along with VST for creating markings [27, 29], but again, this is not DAW-agnostic. It may be possible to create MIDI files or interchange EDL files [27] and import them as markings across several DAWs, but there is not much work in this space yet, and it would take additional significant effort to verify that this works consistently with each DAW. In addition, for both AMT and AS, we need the ability to access multiple audio files at once. VST plugins usually operate on just one audio file, so it would be difficult to build one that works with multiple audio files at the same time. While it is possible, the way this issue is handled is also dependent on the DAW [27]. Requiring a specific workaround for each DAW is in direct opposition to our goal of building something universal, so building VST plugins did not seem like our best option.

Thus, because we wanted AMT and AS to be DAW-agnostic, as well as to have complete access to the environment, we chose to package them as standalone applications. While this meant that AMT and AS would not integrate directly into the DAW, we could still create applications that would integrate well into the sound engineer's workflow. In addition, we also felt that the standalone applications would serve as proofs of concept, that could be extended into tools that integrate directly into the DAW in later versions.

3.2.2 Additional Interviews

We collaborated with three more engineers in the Boston area: Joel Gordon, Antonio Oliart, and Patrick Keating. This section details the interviews and takeaways from them, which helped in determining how to package AMT and AS.

Joel Gordon

Joel Gordon is a freelance recording engineer [6]. Like the engineers at the BSO, he uses Pyramix for recording and producing music. The recording phase of his workflow is completely offline: he does not use his computer and takes notes by hand on the score (using the technique described in Section 2.1.1 of Chapter 2). He is aware of the marking system in Pyramix, but prefers score marking with paper and pencil. He only turns to his computer after recording, to edit and mix, and then uses his score and log of takes to help him choose which recordings to use.

Since Mr. Gordon does not use digital markings, AMT would not be useful for him. We described AS, which would help him search through all recordings for certain snippets of audio. He was interested in such a system since it can take a long time to go through paper logs of takes. We asked how he would visualize using it: ideally, he would want it to be a plugin or part of the DAW that would allow him to input a marking in a "problem section" (one that he wants to replace with a better recording of that part), and then have the DAW return potential replacements of that section. Since we were leaning towards building standalone applications, we also asked if a

separate application would still be useful. Mr. Gordon's response was that it would only be useful if he could batch several problem spots at once (i.e., once he has picked out several problem spots, he goes to the standalone application and deals with them all at once, without having to go back and forth between the DAW and standalone application for every problem spot).

Antonio Oliart

Antonio Oliart is a sound and broadcast engineer at WGBH radio in Boston and also at the BSO (for the live broadcasts) [14]. He uses the Nuendo DAW, which unlike Pyramix, is a linear DAW. This means that the final mix is built up at the end of the timeline, as oppose to in its own track as it is with source-destination DAWs like Pyramix.

While recording, his workflow was similar to that of Mr. Gordon's: he works by marking the score by hand and creates an initial mix from this, and then fishes for better takes via reading waveforms. In addition, he sometimes uses digital markings in Nuendo for sections of pieces, to make it easy to hop around to different takes. Since he uses markings in a different way from the BSO, AMT would not be beneficial for him. However, he was interested in a system like AS, as it would speed up his mixing process by helping to find and choose takes. He said that even a standalone application would be helpful if it gave timestamps of audio matches for some query. Ideally, it could be a plugin that gives a list of matching regions along the timeline. This timeline is one large timeline with all takes, since Nuendo is a linear DAW. Thus, such a plugin would have to be specific to linear DAWs, which was not ideal given our goal of a universal, DAW-agnostic tool. In addition, for the best user experience, the user would need the ability to move the playhead and access the transports of the DAW, and we were uncertain that plugins had such access to these DAW elements.

Patrick Keating

Patrick Keating is another freelance sound engineer who works with live concerts as well as studio recordings (both classical and jazz) [9]. He has used Pro Tools

and Nuendo in the past, and currently uses the Sequoia DAW. Our primary reason to meet Mr. Keating was to learn more about the MuSyc feature in Sequoia (see Section 2.1.3 of Chapter 2).

Mr. Keating's workflow is also similar to that of Mr. Gordon's. He employs the manual marking scheme on score during recording sessions, and sometimes even after listening to all takes. Thus, he has not used the MuSyc feature extensively, because the bulk of his final mix is already mapped out in this way. However, he has used it on occasion to find an extremely specific beat, section of a chord/scale, or ornament, and found it helpful in these instances. He believes it is a useful feature especially because the matching audio takes are stacked one on top of another and synced. This helps in comparing the performance in various takes, as it is fast to find the same point in the music in different takes this way.

From these interviews and our research, it became clear that we should take the standalone application approach. We also learned that while AS would be useful for all sound engineers, AMT would only be used by the BSO engineers at this time. In addition, having the tools directly integrated into DAWs in the future would provide the best user experience: we discuss this further in Chapter 6.

Chapter 4

Implementation

This chapter provides the technical details of the algorithms used and implementation of AMT and AS, from the scripts to standalone Python applications. The associated code can be found at the following Github repository: `https://github.com/smritip/dtw-for-classical-music-prod` [16]. This code is open source, under the GNU General Public License [32], and the repository maintains both the scripts and standalone application executables, with a README detailing how to use each.

In this chapter, we discuss the base technologies and how we fit them to a sound engineer's workflow. We then describe the potential usage of the scripts. We conclude with a section on the final products (the standalone Python applications, packaged and distributed as Windows executables).

4.1 Algorithms

As introduced in Section 2.2, the base algorithm for both AMT and AS is DTW. Section 4.1.1 describes how DTW is used for both AMT and AS, and Section 4.1.2 describes how these algorithms are adapted and extended to function as tools in a sound engineer's workflow.

4.1.1 Base Technologies

AMT

DTW for AMT carries out audio synchronization of one full recording of a piece to another, and then uses the synchronization results (points of musical correspondence between the two recordings) to transfers markings from one recording to the other. As explained in Section 2.2, DTW can analyze and measure the similarity between two time series (derived from the two recordings) which may vary in speed. It does so by "warping" the series in the time domain: for a given musical moment, DTW translates the time in one recording of that musical moment into the time of the second recording for the same musical moment. In this way, the warping path creates the correspondence between the two recordings: it identifies an optimal alignment between the two series, thus synchronizing them. See Section 2.2 of Chapter 2 for the full details of the DTW algorithm.

Our algorithm for AMT is as follows (also diagrammed in Figure 4-1):

Dynamic Time Warping (DTW) for AMT (Audio Synchronization):
Given: Two audio recordings of the same musical piece (X and Y), and one file of markings (for X).
Goal: Transfer the markings from X to Y, and create a file with these new markings (for Y).
Algorithm:

1. Convert recordings to appropriate representations (we chose normalized chromagrams). We use the LibROSA library [1] to load the sound files as NumPy [2] arrays, and then create chromagrams from those. We use the NumPy library to store and manipulate audio data in an array format.

    - Chroma X: $12 \times N$
    - Chroma Y: $12 \times M$

2. Run the DTW algorithm as described in Section 2.2 of Chapter 2. We use the NumPy library for matrix manipulations.

3. Once we have the correspondence points from running DTW (given as a path that is a list of (recording X sample, recording Y sample) points), we can transfer the markings from recording X to recording Y:

   (a) Parse the file of markings (for X) to extract the markings. Each marking is a data point containing a timestamp and a text label. We use the xml library [21] for this.

   (b) Create a list for new markings.

   (c) For each marking:

   - Find the point in the path the marking is closest too (i.e., in looking at each point (X_i, Y_i), use the marking metadata relating to samples to discover which "X_i" it matches with).
   - The matching point (X_j, Y_j) gives us Y_j, the sample where the marking should go in recording Y. Add this new marking to the list.

   (d) From the list of new markings, create a new file of markings (for Y). We use the xml library [21] for this.

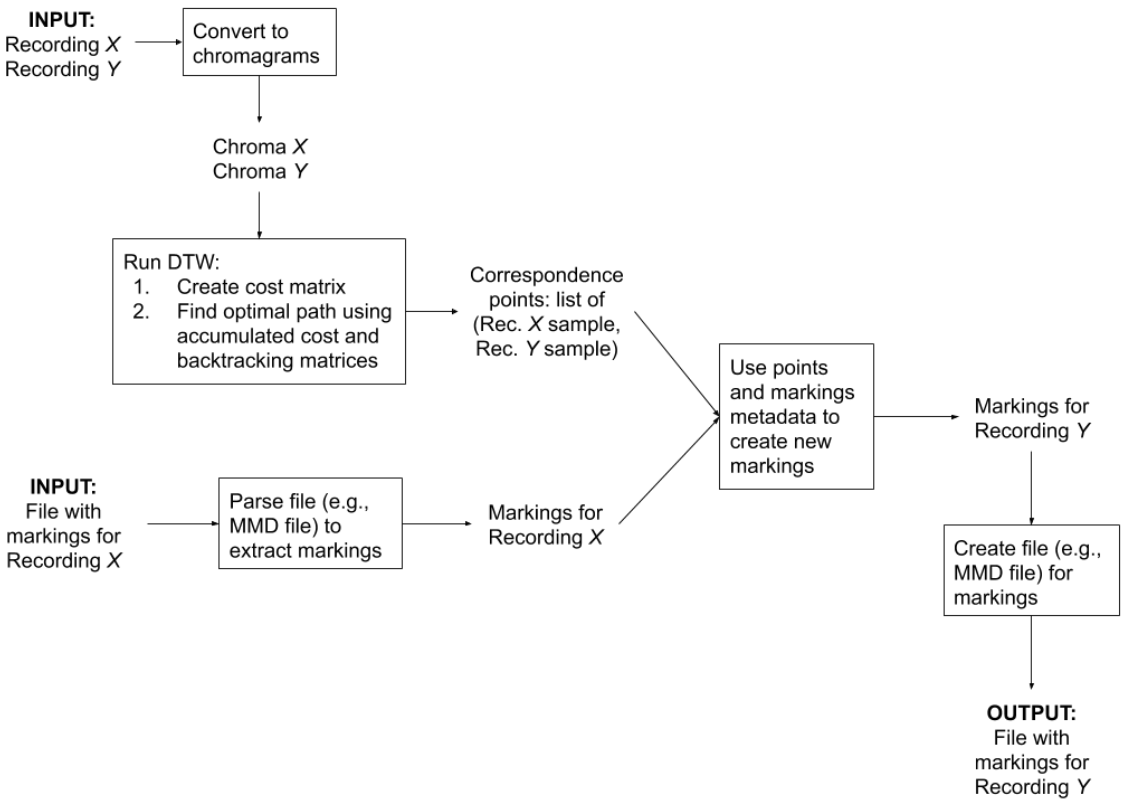

Figure 4-1: Diagram of the AMT algorithm.

AS

DTW for AS carries out audio matching: essentially audio synchronization of one snippet of audio to parts of other recordings (or the whole recording) to find matches. Thus, we run the DTW algorithm described above with the snippet (the query of the search) to other recordings, to find the best match(es). DTW attempts to synchronize the query with each entire recording. Intuitively, the best matches will have the "best" diagonal paths in each of the resulting DTW cost matrices. Our algorithm for AS is as follows (also diagrammed in Figure 4-2):

Dynamic Time Warping (DTW) for AS (Audio Matching)
Given: An audio snippet (the query), other audio recording(s) (forming a database to search through), and the number of matches to find.
Goal: Find the matches in the database.
Algorithm:

1. Convert the query audio to chromagram (we use LibROSA again).
2. For each recording Y in the database:
   (a) Convert the recording to chromagram.
   (b) Run the DTW algorithm (see step 2 in the DTW for Audio Synchronization algorithm).
   (c) The points in the last row of the accumulated cost matrix D (i.e., $D(N, m)$ for all m in $[1, M]$) give the accumulated costs of the optimal warping paths that end at $D(N, m)$. We call this last row the matching function.
   (d) Find the best match by finding the lowest cost (argmin) from the matching function:
      - For each match, we need the beginning and ending timestamps (for the user to identify the matches). The matching function corresponds to the end of the matches: this gives us the ending timestamp for the

match. Use the backtracking matrix B to find the beginning of each match: this gives us the beginning timestamp for each match.

(e) To find more matches, repeatedly squash out the surrounding area and find the argmin (and beginning and ending timestamps) until the desired number of matches have been found. The matches can be ordered from the best match to the least by comparing the accumulated costs of each path: the lower the cost, the better the match.

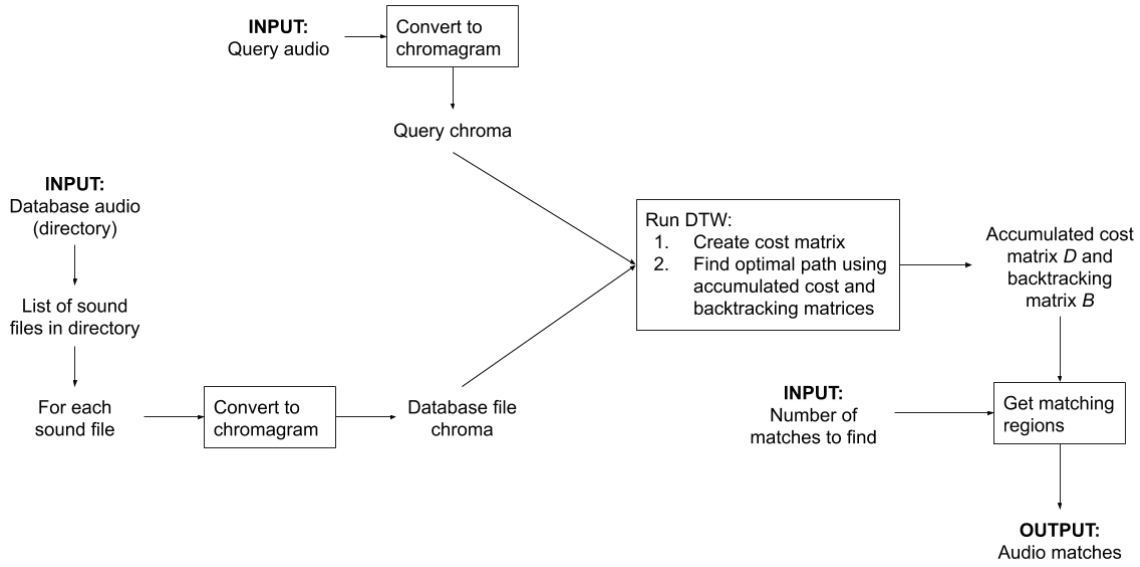

Figure 4-2: Diagram of the AS algorithm.

4.1.2 Fitting to Workflow

The previous section described the base algorithms which should work in any environment and workflow, with extensions and/or adaptations (e.g., working with different formats of markings). This section describes the technologies built on top of the base algorithms to fit AMT and AS into a sound engineer's workflow.

AMT

As we found in our research, at this time, AMT will only be useful for the BSO engineers (among our cross-section of surveyed engineers), since they are the only ones who use markings heavily and would find it helpful to transfer them from recording to recording. Thus, the adaptations for AMT were to make the base algorithm work with Pyramix and BSO recording protocol.

Pyramix works with markings ("MediaMarker" is the proprietary name) via MMD files, which is a type of XML format [27]. Thus, to parse old markings and create new markings (needed as part of AMT), we wrote an MMD parser and creator. The parser walks the XML tree and extracts the relevant marking information (e.g., the rehearsal number). The creator, given marking information, creates an XML tree and MMD file that Pyramix can interface with.

Each marking has at the minimum a unique identifier (UID) entry and a timecode entry. The UID is unique to Pyramix and is a 16 byte dash-delimited string of random bytes (random integers from 0-255). In creating new markings, we wrote a function to generate new UID entries that followed these requirements.

The timecode entry is also unique to Pyramix, and represents the time from the start of the recording. There is a multiplier used to calculate the number of samples from the timecode, which is dependent on the sampling rate. The sampling rate that the BSO uses is 96kHz and the corresponding timecode multiplier is $\frac{1}{23520}$. To convert the timecode to samples, multiply by the timecode multiplier (e.g., timecode $5033164800 * \frac{1}{23520} \approx 213995$ samples). To convert the samples to time in seconds, divide by the sample rate (e.g., $\frac{213995 samples}{96000} \approx 2.23$ seconds).

These constants are maintained in a file in the repository, and we use these constants in both parsing and creating new markings, since they inform where markings are in a piece [1].

[1] Each time we interface with the MMD file (to parse or create) we first convert to time before samples, since time is a global invariant across all sampling rates.

AS

AS did not need to be tailored to any specific workflow or DAWs, since, unlike AMT, it does not assume nor require anything of the work environment or DAW.

4.2 Scripts

The Python scripts created as prototypes for AMT and AS can be used from the command line as tools, if an engineer does not want or need to use the standalone applications and their GUIs. Instructions on how to use each of the scripts are included in each script file.

4.3 Standalone Python Applications

Once we had working scripts for AMT and AS, we focused on providing a good user experience. Not all sound engineers may be familiar with the command line to run the scripts, and we wanted to create a GUI for AMT and AS, so we developed applications with a simple GUI for each tool. This part of the development process was the most iterative, in terms of creating a GUI and iterating on it given feedback from the sound engineers. The final GUIs, feature set, and design choices (as of August 2019) are presented in Section 4.3.2.

4.3.1 Framework

We utilized the PySimpleGUI framework [4] to provide GUIs for AMT and AS. PySimpleGUI is a wrapper for Tkinter, Qt, WxPython, which are interfaces to Python's de facto standard GUIs [4, 22]. PySimpleGUI allowed us to create custom GUIs.

For these tools, we created simple applications that would take in the necessary user input, run the scripts in the background, and then return the results, in a user-friendly GUI. To do so, for each application, we defined a layout of user input boxes

and buttons to fit the requirements for each tool and then run the scripts. For a better user experience, we also added threading to the applications, running the GUI in one thread and the AMT/AS scripts in another, so users could continue to interact with the GUI even as AMT/AS runs.

In addition to the libraries used in the AMT and AS scripts, we used the Pygame library [3] for sound output (e.g., playing audio) and Python's threading and trace modules to make the applications multi-threaded [12].

4.3.2 Final Products

AMT

AMT carries out the transfer of markings from one recording to another. Thus, in order to transfer the markings from Recording X to Recording Y, the user must provide the path to these sound files. AMT will also look for the MMD corresponding to Recording X, but we do not need the user to supply that because the assumption is that it will be the same exact path as the recording, just an MMD file instead of a sound file (e.g. the sound recording "/home/audio/recording.wav" will have the corresponding marking file "/home/audio/recording.mmd"). Pyramix (at this time AMT is only useful to Pyramix users) requires this naming convention of sound (i.e., "wav") files and their corresponding MMD files [27], so we did the same. Once complete, AMT will create another MMD file (one for Recording Y) at the same location on disk.

Figure 4-3 below shows what the AMT app looks like upon opening it. The features of the application are (numbers correspond to those in Figure 4-3):

1. User input boxes for paths to wav files.

2. "Browse" buttons to select a file instead of having to type out full paths.

3. "Transfer" button to initiate the AMT process.

4. "Cancel" button to cancel the AMT process at any time.

5. "Close Window" button to stop and close the application at any time.

6. A progress bar to show the progress of the AMT process when running (shown in Figure 4-4 below).

These features are only usable when there are valid user inputs given (e.g. "Transfer" is only clickable when two valid wav files have been supplied).

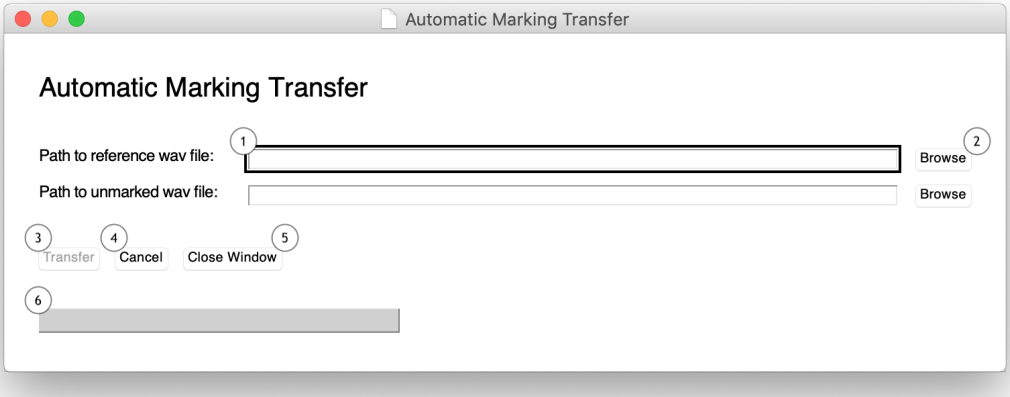

Figure 4-3: GUI for AMT. Features (numbered 1-6) are described in Section 4.3.2.

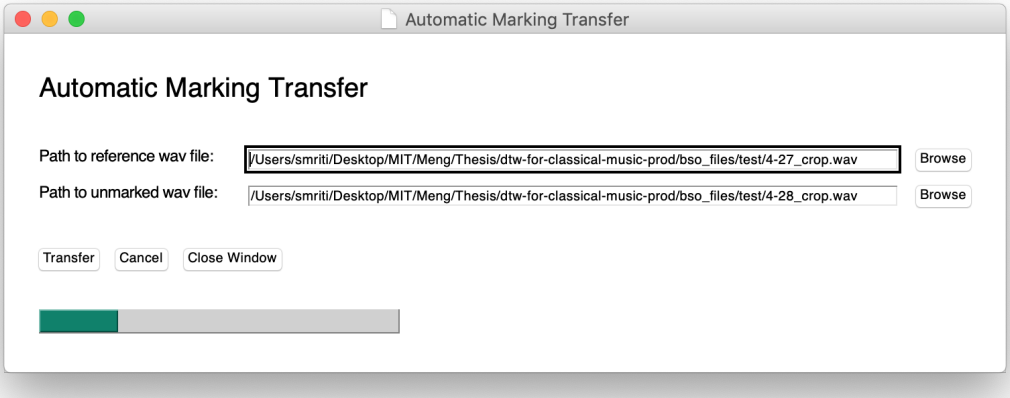

Figure 4-4: AMT in use.

AS

AS carries out audio matching, specifically of one search query to a database of audio. Thus, the user must provide a path to the query audio, a path to audio files to search through, and the number of matches to search for. In addition to these basic requirements, AS also provides a media player, to listen to any of the audio clips. Once a search is complete, AS will display the results (which may also be loaded into the media player).

Figure 4-5 below shows what the AS app looks like upon opening it. The features of the application are (numbers correspond to those in Figure 4-5):

1. User input boxes for paths to query audio, audio files, and number of matches.
2. "Browse" buttons to select a file/folder instead of having to type out full paths.
3. "View Query" button which loads the query into the media player and allows for playback, so the user can confirm their query. See Figure 4-6 below.
4. Start time and end time user inputs for the query audio, so the user can select any portion of a wav file as a query. See Figure 4-7 below.
    - If these are blank, AS uses the whole wav file as the query.
5. "Search" button to initiate the AS process.
6. "Cancel" button to cancel the AS process at any time.
7. A progress bar to show the progress of the AMT process when running. See Figure 4-8 below.
8. Matches section to display the matches. See Figure 4-8 below.
9. "View" button for each match to load the audio matches into the media player and allow for playback, so the user can listen to the results. See Figure 4-8 below.

10. "Prev" and "Next" buttons to browse through all match results, if there are more than five. See Figures 4-8 and 4-9 below.

11. Media player to load audio, display the waveform, and allow for playback (including play, pause, and rewind buttons). Also includes a "Now Playing" section to display the currently loaded audio. See Figure 4-6 below.

12. "Close Window" button to stop and close the application at any time.

These features are only usable when there are valid user inputs given (e.g. "View Query" is only clickable when a valid wav file has been supplied).

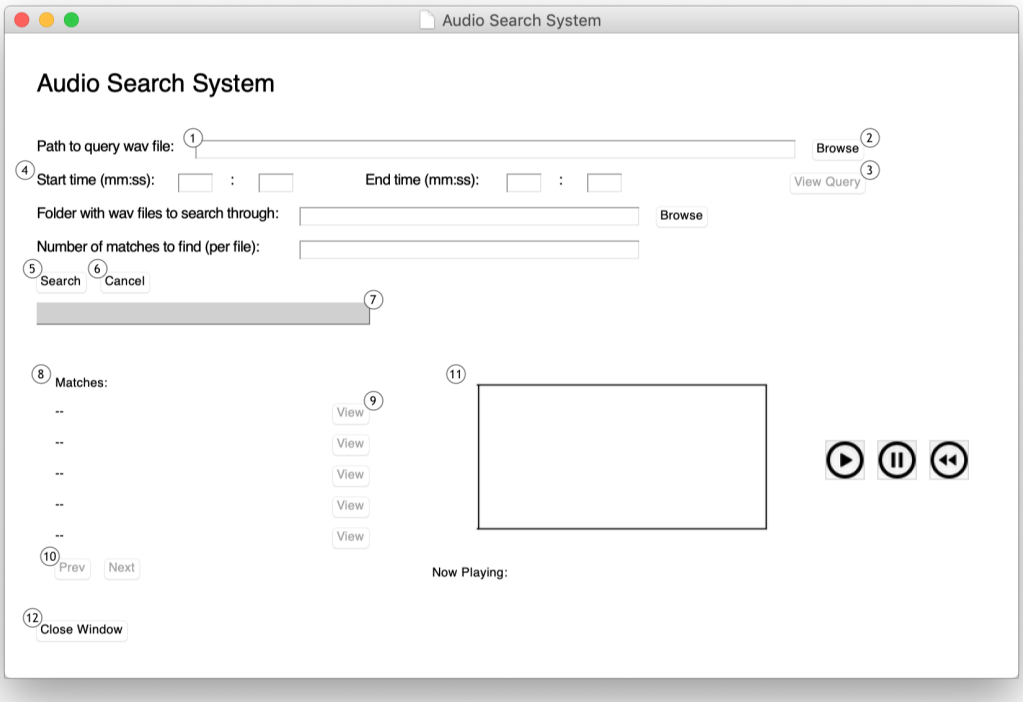

Figure 4-5: GUI for AS. Features (numbered 1-12) are described in Section 4.3.2.

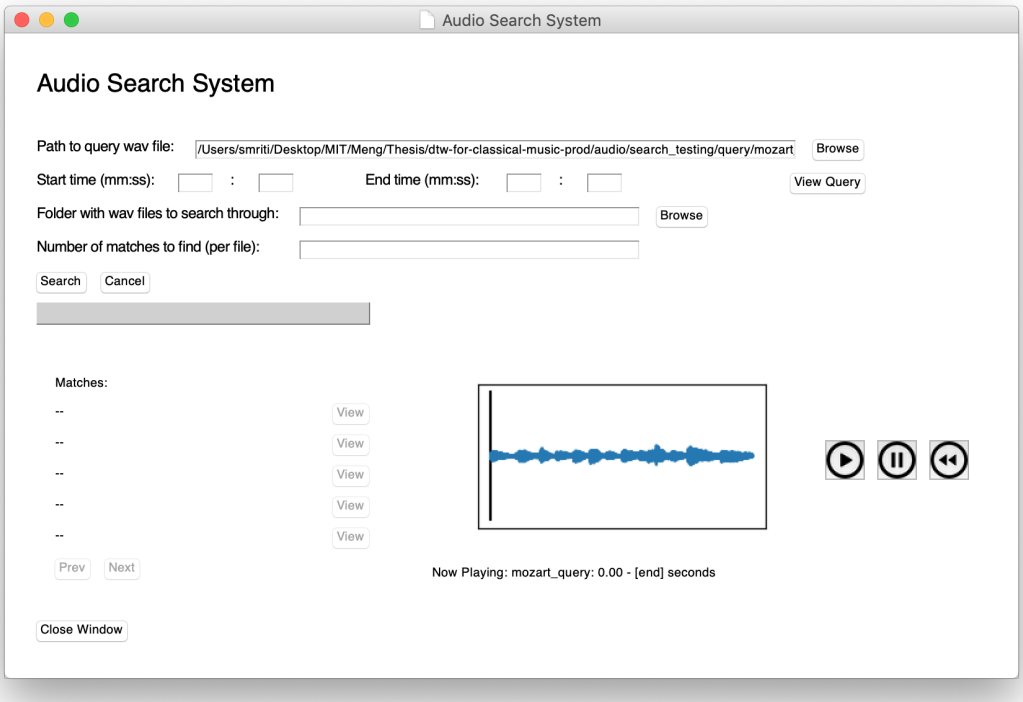

Figure 4-6: Query audio loaded into media player.

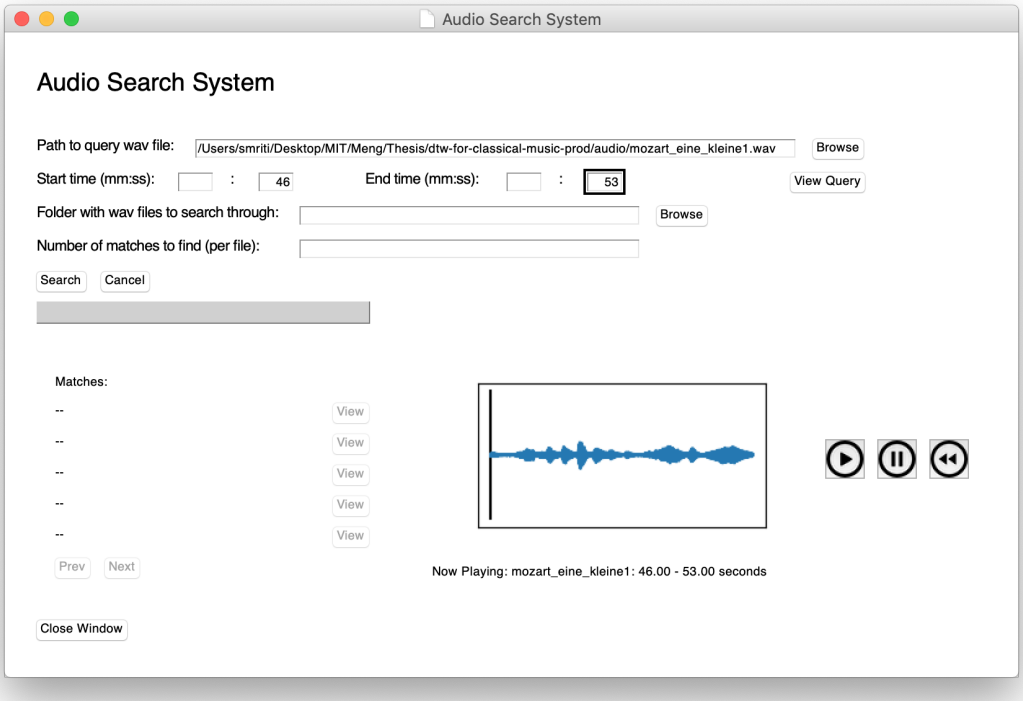

Figure 4-7: User may supply start and end times to select a portion of audio for the query.

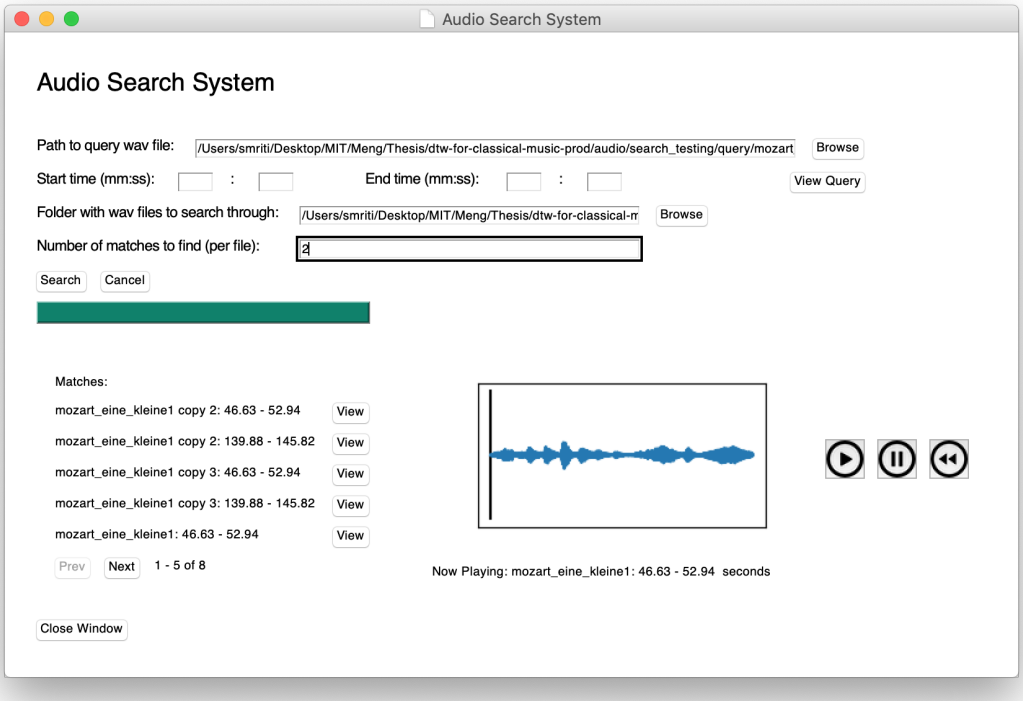

Figure 4-8: After AS completes, the matches section is filled, progress bar is complete, and the user may load any match (or the query again) into the media player. This example shows matches 1-5, and the 5th match playing.

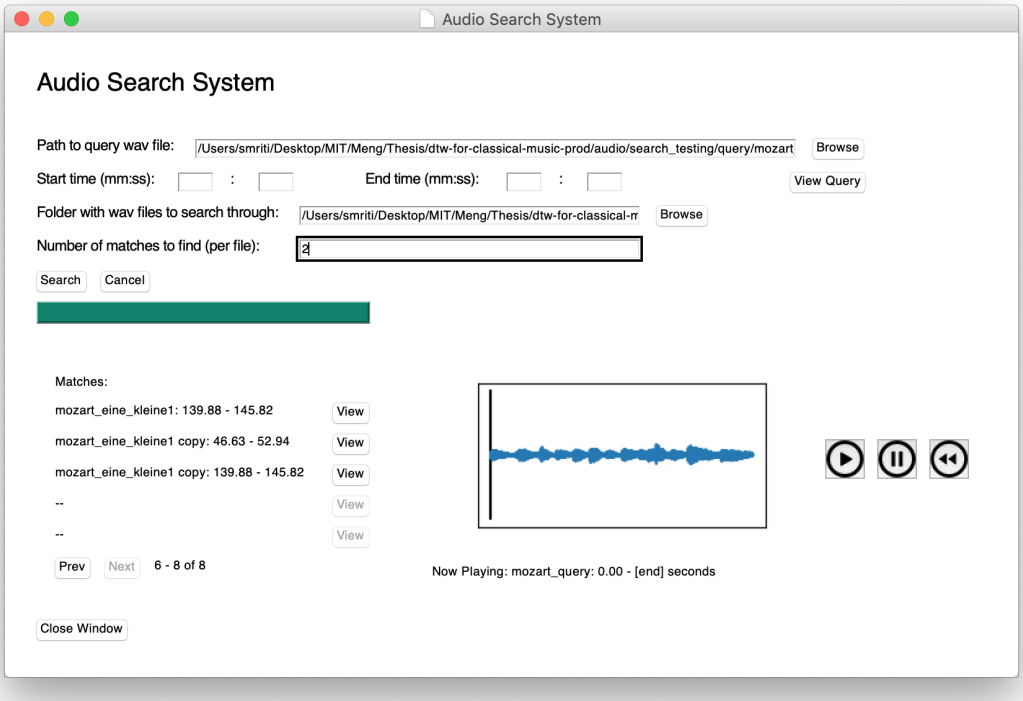

Figure 4-9: After AS completes, the matches section is filled, progress bar is complete, and the user may load any match (or the query again) into the media player. This example shows matches 6-8, and the query playing.

4.3.3 Creating Windows Executables

If a user has Python 3 installed, and all libraries that these tools use as well (see Section 4.3.4), they can run the applications from the command line. For example, to run the AMT app, they could run the following:

```
>> python3 amt_app.py
```

However, we wanted any engineer to be able to use these tools, including engineers that do not have Python 3 and the necessary libraries installed and/or may not be comfortable using the command line. Thus, since the majority of engineers work on Windows machines [27], we decided it would be best to package and distribute the Python applications as Windows executables. These executables do not require the user to have Python 3 or any of the libraries used installed [19].

To do this, we explored several options: PyInstaller [18], py2exe [17], nuitka [7], and PyOxidizer [31]. We attempted following tutorials and reading documentation for each. Py2exe does not support all versions of Python 3, and we were unsuccessful in building any executables using nuitka. PyOxidizer seemed promising, but as of July 2019, did not have support for C extension modules, which our tools utilize (e.g., PySimpleGUI, tkinter). Thus, we chose to use PyInstaller. There is one known issue with using PyInstaller with LibROSA (dependency packaging issue) [8], a library used in both AMT and AS, so we rewrote versions of the AMT and AS applications that use the wavfile library (from scipy.io) [26] instead of LibROSA, in order to create Windows executables using PyInstaller. The Windows executables for the AMT and AS applications can be found in the Github repository.

The executable for the AS application currently does not include the media player. This is because the media player heavily uses LibROSA, and wavfile does not include all of the functionality that LibROSA provides that we need (e.g., seeking in a track efficiently), so the application would be too inefficient and run too slowly. However, as discussed in Chapter 5, the BSO engineers did not believe the media player was a

necessary feature, so the executable should still be useful even without it. In addition, engineers can still use the AS application with the media player via the commandline, as described in the beginning of this section.

4.3.4 Python Libraries Used

In implementing these tools, we used several libraries. These libraries have been discussed previously throughout this chapter; this section provides, for reference, a full list of all the libraries used, along with their use case(s) in our work:

- LibROSA [1] for MIR and audio analysis [used in AMT and AS scripts]
- NumPy [2] for matrix manipulation
- xml [21] to parse and create XML (specifically, MMD) files
- Pygame for sound libraries [3] [used in AS's media player]
- threading [20] for building multithreaded applications [used in AMT and AS applications]
- trace [12] for killing threads [used in AMT and AS applications]
- wavfile (scipy.io) [26] for MIR and audio analysis [used in versions of AMT and AS applications to create Windows executables [2]]

[2] As mentioned in Section 4.3.3, we could not use LibROSA with PyInstaller to create Windows executables, so we had to rewrite the applications using the wavfile library.

Chapter 5

Evaluation and Results

We used three metrics to evaluate AMT and AS:

1. Accuracy,

2. Time saved for the sound engineer, and

3. Ease of use and integration into the engineer's traditional workflow.

At this time our evaluation has been primarily qualitative, and limited to testing and interactions with the BSO. As development was largely iterative, we used their feedback to incorporate changes and improve the products.

To create the best products, one of the primary focuses of work in the future will be to continue with more rigorous testing and analysis, which will require more audio test data as well as human testers. We go into this in more detail in Section 6.1 of Chapter 6.

This chapter details the results from our evaluation of AMT and AS.

5.1 AMT

Before testing AMT with BSO engineers, we unit tested the software to verify the functionality was correct. Then, we carried out real tests with actual recordings from the BSO. The setup was as follows: we had access to two recordings (Recording X,

Recording Y) of the same piece (recorded by the BSO) and two corresponding MMD files (MMD X, MMD Y) of markings. We supplied Recording X, Recording Y, and MMD X to the AMT script, which transferred the MMD X markings from Recording X to Recording Y, and in turn created a new MMD file (MMD NEW). We expected MMD NEW to have the same markings as MMD Y (the ground truth), and indeed it did. The response from Mr. Watts at the BSO was:

> "Your test MMD is spot on, I think the biggest difference I calculated was 228ms ... which is well within the margin of error. I'm almost positive I can't even get them that close manually. I'm very impressed!" [27]

In terms of user experience, AMT will drastically decrease the amount of time and manual work in preprocessing recordings for mixing. Mr. Squire told us that AMT will help them save four to five hours per musical piece they produce (the time it takes to mark all recordings for a piece), which, in the course of the year if they do about ten recordings, could save them 40-50 hours [27]. To make most efficient use of AMT, Mr. Squire and Mr. Watts have requested a few more features for the application in the future, which are described in Section 6.3 of Chapter 6.

5.2 AS

In contrast to AMT, there are no preexisting "ground truths" to verify the AS tool. Instead, we chose a piece of classical music (Mozart's "Eine Kleine Nachtmusik" [37]), which had several repeated phrases to verify the accuracy of the algorithm and later tool. After we verified it worked in this setting, we needed to make sure it would work with real data (which could be longer, be more complicated, or have noise), so we requested some data from Mr. Oliart and used the AS app to search for matches. While AS successfully found matches in this particular test case, it has to be subjected to more extensive and rigorous functional testing. Since AS is very compute-intensive, we also need to carry out performance testing for it under varying CPU/memory load conditions, from light to very heavy, as well as with different sound file sizes (due to differences in recording quality and/or length).

To evaluate user experience, we involved the BSO engineers in testing different phases of the AS app, to understand what features were useful and if the GUI was intuitive. This led to a final version of the app that fits well into the workflow, since its design was heavily informed and influenced by the users. One interesting piece of feedback was that the BSO engineers may not use the media player for logistical reasons [1]. However, they recommended keeping the feature as it may be useful to other sound engineers with different setups [27].

[1]When working on their machines with Pyramix, the only audio inputs and outputs are through Pyramix, so other audio sources, like the AS application, would not be heard.

Chapter 6

Future Directions

This chapter discusses several future directions for the work described in this thesis: further evaluation and testing, the distribution of AMT and AS, additional features and applications of AMT and AS, and more potential tools.

6.1 Further Evaluation and Testing

AMT and AS will benefit from additional qualitative and quantitative evaluation. So far, we have tested our software with unit tests and real data, and we have observed and received feedback from engineers at the BSO. We would like to send more users AMT and AS and follow up with surveys (open response and numerical scale questions) and carry out more A/B testing to compare workflows before and after incorporation of AMT and/or AS. This will also help inform the best way to integrate AMT and AS into engineers' workflows in future versions.

6.2 Distribution of AMT and AS

Currently, AMT and AS are available (open source) for use as Python scripts, Python applications, and Windows executables on Github [16]. From our interviews with potential users, we understood the convenience of using a tool that is part of the native DAW. While we can explore newer plugin formats like VST3 [29], we would

like to reach out to DAW developers and see if we can integrate AMT and/or AS into their DAWs, as parts of their system. We are already in conversation with Merging Technologies, the creators of Pyramix [33], who are potentially interested in such a collaboration. If successful, AMT and AS would become features packaged along with the Pyramix distribution.

6.3 Additional Features

While developing AMT and AS, we received feature requests and kept track of more potential features that are not yet part of the tools. The following is a list of additional features to consider implementing and including in the tools:

Applicable to both AMT and AS:

- Create a Mac application (e.g., .app), which will not require the install of Python and other libraries used.
  - py2app [15] is a potential library to use to package and distribute the tools for Mac
- Make the tools faster and more robust; possible areas to explore include:
  - Optimizations and approximations of DTW algorithm (e.g., FastDTW [25], Sakoe-Chiba bound [24])
  - Investigating other feature representations of audio (e.g., spectral features, chroma difference features)
  - Rewriting the tools in C++ and using JUCE [23] to build C++ applications,
  - Alternative approaches to DTW for audio matching (e.g., diagonal matching method [13])
  - PySimpleGUI: At this time, we used PySimpleGUI for simplicity and for a first version of the tool. However, if AMT and AS remain standalone

applications, it would be worth it to look into other options for building a GUI.

  - Multithreading: Explore more ways to implement threading, in hopes of making it more efficient.

AMT

- Multiple transfers: Instead of a transfer of markings of Recording X to Recording Y, allow for a transfer of markings of Recording X to several Recordings Y, Z, and so on.

- Allow usage of pmf files: Pyramix currently works with pmf files, not wav files. Currently, the BSO engineers convert the pmf files into wav files (since they are different audio formats) before using AMT. Do this conversion as part of AMT, thus allowing AMT to work directly with pmf files.

- Partial marking transfer: For AMT, we considered doing not just full to full recording synchronization, but also marking smaller sections of pieces. We realized that for classical music, however, this did not make sense. This is because classical music typically has repetitions so a smaller section of a piece may match to several places in the full recording, and we would not know how to choose between the several, equally-likely options.

AS

- Search via markings: if the query and database audio have markings, use that metadata information instead of analyzing the audio content.

- Volume control for media player

- Interactive media player (e.g., moving the playhead in the player)

- A Windows executable for the application that includes the media player (see Section 4.3.3 of Chapter 4)

6.4 Other Applications of AMT and AS

While we focused on AMT and AS for classical music, it would be interesting to explore such tools in other music realms. While the workflow and musical structures are different for every genre, it is still worth exploring to see if AMT and/or AS could fit (e.g. pop music has some structure with chorus/verses, and jazz with theme and variations [27]). If these particular tools are not applicable, one could also look into what other tools may be helpful in those realms.

In addition, AMT and AS may be valuable outside of the music production world. For example, in the Automated Dialog Replacement (ADR) and video game fields, engineers record a lot of the same audio multiple times. Thus, these tools could help engineers in those fields identify audio content and find audio matches in their editing and production processes [27]. In these cases, the input representation will likely be different from what we use in our implementations of AMT and AS: instead of a chromagram representation, such applications would use representations that are more tuned to the requirements of the particular audio (e.g., ADR works with speech, which is different from classical music).

6.5 More Potential Tools

This is just the first step in bridging the gap between music production and DSP/MIR. Additional research will lead to the discovery and creation of other useful tools.

In addition, while we focused on how DTW could help the process, there exist more DSP/MIR algorithms that may help. One specific example is that the diagonal matching method (introduced in Section 6.3) could be utilized for audio matching, instead of DTW. More generally, MIR algorithms involving applications like beat-tracking, automatic chord detection, and automatic section detection [13], may also be useful in DAWs and music production, even outside of the classical musical realm (i.e., for other genres like pop, rock, and jazz). Investigation of more algorithms may open up other worlds of potential tools.

Chapter 7

Conclusion

In this thesis, we explored areas of the classical music production workflow that could benefit from DSP and MIR-based tools. One of the biggest pain points of this production process is identifying the content of several tracks of music while editing and mixing. This led us to identify and formulate requirements for two tools: Automatic Marking Transfer (AMT) and Audio Search (AS).

While a large part of this thesis was identifying pain points of the classical music production workflow, collecting requirements for tools to address these pain points, and then implementing the underlying algorithms and developing the tools, it was equally important to us to make sure that the tools fit seamlessly into a sound engineer's workflow and improved their overall production experience. To work towards this latter goal, we made design choices to ensure that the tools could be used universally (i.e., were DAW-agnostic), and we did not assume anything about the engineer's environment (i.e., did not require any prior installations): we created standalone Python applications and packaged and distributed them as Windows executables. We received positive feedback on both the accuracy and user experience of the AMT and AS tools, though we would like to continue further evaluation.

There are several future directions we would like to explore: in addition to more evaluation of AMT and AS, we would like to partner with DAW developers and see how we may integrate AMT and AS into their systems. We would also like to improve on and add more features to AMT and AS, and explore other applications of these

tools.

AMT and AS serve as proof that we can bring the worlds of music production and DSP/MIR together to create successful, useful tools that improve the classical music production workflow. There are additional areas in the production process to explore: we can create more tools and use different algorithms. We are optimistic about the possibilities and the future of such creations, and we look forward to the community innovating and collaborating in this interdisciplinary area.

Bibliography

[1] LibROSA. https://librosa.github.io/librosa/, 2019.

[2] NumPy. https://numpy.org/, 2019.

[3] pygame.mixer. https://www.pygame.org/docs/ref/mixer.html, 2019.

[4] PySimpleGUI. https://pysimplegui.readthedocs.io/en/latest/, 2019.

[5] Avid. Pro Tools. https://www.avid.com/pro-tools, December 2018.

[6] Joel Gordon and Smriti Pramanick. Interview with Joel Gordon, Freelance Recording Engineer., May 2019.

[7] Kay Hayen. What is Nuitka. https://nuitka.net/pages/overview.html, 2019.

[8] GitHub Issues. problems with pyinstaller. https://github.com/librosa/librosa/issues/538, 2017.

[9] Patrick Keating and Smriti Pramanick. Interview with Patrick Keating, Freelance Sound Engineer and Sequoia User, May 2019.

[10] MAGIX. *Sequoia 13*. MAGIX Software GmbH, 2014.

[11] MAGIX. Sequoia 15: The Engineers' Choice. https://www.magix.com/us/music/sequoia/classical-productions/#c735079, 2019.

[12] Manthanchauhan. Python — Different ways to kill a Thread. https://www.geeksforgeeks.org/python-different-ways-to-kill-a-thread/, January 2019.

[13] Meinard Müller. *Fundamentals of Music Processing*. Springer, 2015. Audio, Analysis, Algorithms, Applications.

[14] Antonio Oliart and Smriti Pramanick. Interview with Antonio Oliart, Sound and Broadcast Engineer at WGBH radio in Boston and at the Boston Symphony Orchestra (for Live Broadcasts), May 2019.

[15] Ronald Oussoren and Bob Ippolito. py2app - Create standalone Mac OS X applications with Python. https://py2app.readthedocs.io/en/latest/, 2012.

[16] Smriti Pramanick. smritip/dtw-for-classical-music-prod. https://github.com/smritip/dtw-for-classical-music-prod, 2019.

[17] py2exe. py2exe. http://www.py2exe.org/, February 2019.

[18] PyInstaller. PyInstaller. https://www.pyinstaller.org/, July 2019.

[19] PySimpleGUI. Creating a Windows .EXE File. https://pysimplegui.readthedocs.io/en/latest/cookbook/#creating-a-windows-exe-file, 2019.

[20] Python. threading Thread-based parallelism. https://docs.python.org/3/library/threading.html, July 2019.

[21] Python. XML Processing Modules. https://docs.python.org/3/library/xml.html, July 2019.

[22] Kenneth Reitz and Real Python. GUI Applications - The Hitchhiker's Guide to Python. https://docs.python-guide.org/scenarios/gui/#pysimplegui, 2019.

[23] ROLI. JUCE. https://juce.com/, 2019.

[24] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE*, 26(1):43–49, 2 1978. Transactions on Acoustics, Speech, and Signal Processing.

[25] Stan Salvador and Philip Chan. Fastdtw: Toward accurate dynamic time warping in linear time and space. *Florida Institute of Technology*, 2004.

[26] SciPy.org. Input and output (scipy.io). https://docs.scipy.org/doc/scipy/reference/io.html, May 2019.

[27] Nick Squire, Joel Watts, and Smriti Pramanick. Series of Interviews with Nick Squire and Joel Watts, the Boston Symphony Orchestra's Chief Recording Engineer and Associate Audio Engineer, respectively, November 2018.

[28] Icon Staff. 5 FREE VST PLUGINS: JULY 2018. https://iconcollective.com/free-vst-plugins-july-2018/, July 2018.

[29] Steinberg. VST3: New Standard for Virtual Studio Technology. https://www.steinberg.net/en/company/technologies/vst3.html, July 2019.

[30] E-Home Recording Studio. Music Production 101: The 4 Steps to Recording a Song. https://ehomerecordingstudio.com/how-to-record-a-song/, October 2018.

[31] Gregory Szorc. PyOxidizer. https://pyoxidizer.readthedocs.io/en/latest/, 2019.

[32] Merging Technologies. GNU General Public License. https://www.gnu.org/licenses/gpl-3.0.en.html, 2007.

[33] Merging Technologies. Pyramix 11. https://www.merging.com/products/pyramix, December 2018.

[34] Wikipedia. Digital audio workstation. https://en.wikipedia.org/wiki/Digital_audio_workstation, December 2018.

[35] Wikipedia. Virtual Studio Technology. https://en.wikipedia.org/wiki/Virtual_Studio_Technology, November 2018.

[36] Wikipedia. Chroma feature. https://en.wikipedia.org/wiki/Chroma_feature, February 2019.

[37] Wikipedia. Eine kleine Nachtmusik. https://en.wikipedia.org/wiki/Eine_kleine_Nachtmusik, July 2019.

[38] Wikipedia. MIDI. https://en.wikipedia.org/wiki/MIDI, August 2019.