Automated Music Transcription

Michael Bell

Table of Contents:

Background 2

Approach 4

- Number of Notes 4
- Maximum Intervals 6
- Tied Notes 8
- Post Processing 10

Merits 12

Challenges 14

Limitations 17

Future Work 20

Background

The Automated Music Transcription project is an ongoing undergraduate research topic supervised by Dr. Papamichail. Its overall purpose is to create a piece of software using the Music21 Python library that takes an existing musical score and arranges it for a new set of instruments, ensuring the playability of each individual part while keeping the output as faithful to the original sound as possible. Previous students have made significant progress towards this goal, with projects focused on music arrangement, voice separation, and reduction. Research in music arrangement focused primarily on taking a set of monophonic parts and finding the best arrangement for a set of monophonic instruments (monophonic meaning instruments or parts that can only play a single note at any given time; cannot play multiple notes at once). The algorithm checks different arrangements to determine which makes the most sense based on the set of instruments by looking at the range of notes each instrument can play, transposing the entire score up or down if necessary. Research in voice separation focused on taking polyphonic parts (parts that play more than one note at a time) and separating them into multiple monophonic ones. Emphasis is placed on ensuring each part makes musical sense and on figuring out which instruments should drop out or join in if the number of notes in the original part changes. Finally, research on reduction focuses on taking a polyphonic part and reducing the number of notes played at a time to a given maximum. For example, if the maximum is 1, the part will be reduced to a monophonic part. The biggest focus in this research is searching for and storing patterns to aid in keeping the melody intact and keeping the most important harmonies when reduction is performed in order to ensure integrity of the original score. When chained together, the research done in these three areas allows a user to take any score, reduce it to a maximum number of notes, split the result into any number of monophonic parts, and arrange these parts to

any set of monophonic instruments, with each step of the process keeping their respective output as faithful to their input as possible.

As far as monophonic instruments are concerned, the project is already a success when the existing algorithms are used in sequence. However, many popular instruments, such as the piano and the guitar, are polyphonic. Monophonic instruments' biggest noticeable constraint (other than being monophonic) is the range of notes that each can or cannot play, where polyphonic instruments introduce additional challenges and unique constraints. Checking whether the notes of a part fall within a certain range is no longer sufficient to validate whether a part is possible for a polyphonic instrument, and as a result, determining whether a passage is physically possible to play becomes a nontrivial component to our automated arrangement system.

My contribution to the Automated Music Transcription project begins to address the problem of polyphony by creating a piano validation algorithm that determines whether a given part is possible for a single pianist to play on the piano. Given an existing musical score, my algorithm seeks to provide a boolean output of whether the score is possible to play as is, without making any adjustments to the piece, such as removing notes or parts. It then additionally produces a new .musicxml file containing a piano arrangement of the score, split into intuitive left- and right-hand parts, highlighting any chords that are impossible to play. Each polyphonic instrument has its own unique set of constraints, so my work aims to bridge the gap between the monophonic research we already have and the complete polyphonic validation algorithm we will one day have in the future.

Approach

The first step of creating the piano validation algorithm was to figure out the unique constraints of a piano. I spent some time sitting at a keyboard and playing large chords in order to study my own hands, paying attention to what made a chord physically possible or impossible to play. I determined that three things prevented chords from being possible: 1) the total number of notes in the chord, 2) the interval between each note (interval meaning distance between two notes), and 3) tied notes locking certain fingers in place. After using code from previous students to set up the environment, I started implementing my findings into the program.

Number of Notes

In order to find the number of notes played at any given time, my program must first combine every part of the original score into a single part in order to make things simpler for the algorithm. This is done with the chordify() function built into the Music21 library. The result can often consist of large chords spanning much of the piano, illustrated in the figure below.

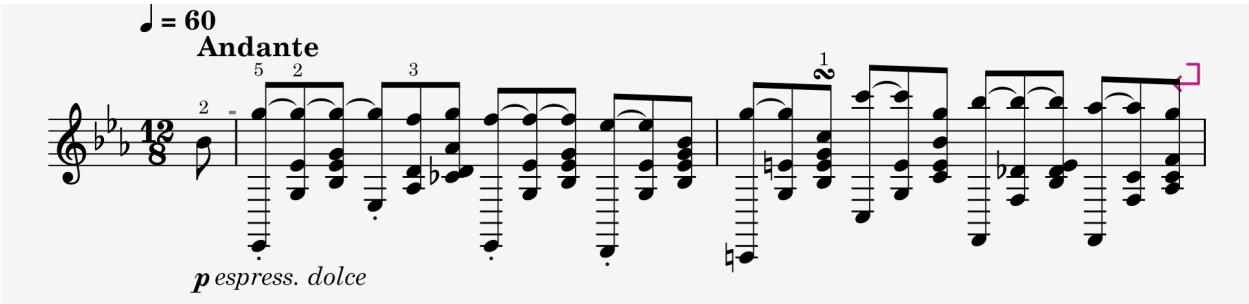

Figure 1 -
chordify() function used on Nocturne Op.9 No.2 by Frederic Chopin.

Using this newly created part, the program goes through each chord one by one from beginning to end. It first finds the largest interval between two notes as the ideal split point between the

left- and right-hand parts, then iteratively moves the split point up or down by a single note until the proper conditions are met.

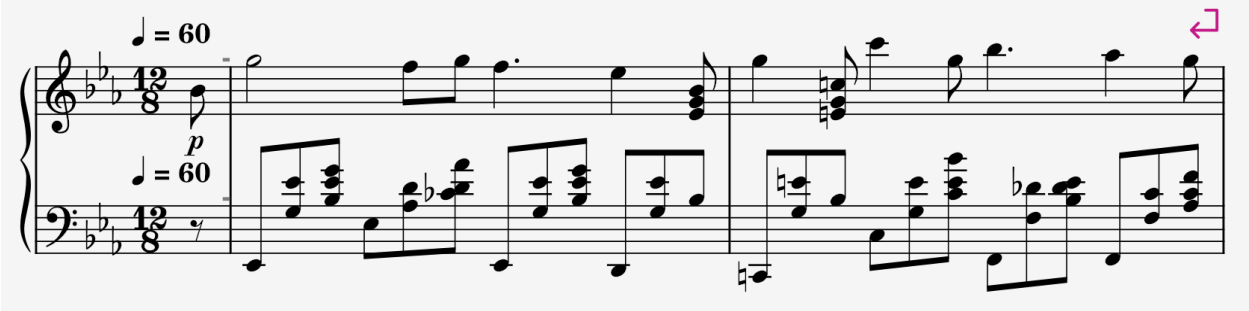

Figure 2 -
Nocturne split into left- and right-hand parts, nearly identical to the original left/right split.

At first, I naively assumed that the maximum number of notes played by one hand was five–one for each finger. Through testing, I realized that some chords deemed impossible were in fact possible to play. This is because each finger is actually capable of playing more than one note in some circumstances. When a finger is placed on the gap between two white keys, it can easily and consistently play both at once. The thumb can also do the same with black keys. In order to determine how many fingers are necessary to play a chord, notes are checked in order starting from index 0, and if the next note abides by the rules explained previously, the next index is skipped. Each index checked is appended to a list. After the loop reaches the end of the chord, the index list will contain every location a finger is actually needed. Using this list, we can more accurately determine whether a chord needs more than five fingers to play, now with an absolute maximum of 10 notes per hand before it is assumed impossible; if any chord of the piece goes beyond the limit of five fingers per chord, the piece as a whole is considered unplayable by a single pianist.

Maximum Intervals

While testing, I found that my own thumb and pinky fingers could play a maximum interval of 16 semitones, so after checking the number of fingers needed in a chord, my program also checked the distance between the lowest and highest note to make sure they were not too far apart from each other. However, it was another naive assumption that this was the only necessary interval constraint, as this let some impossible chords go undetected. For example, if one were to stretch their thumb and pinky fingers as far away from each other as possible, a gap forms between the thumb and the index finger where no notes can be played. It turns out that each pair of fingers has their own limit of how far they can stretch apart. While my thumb and pinky fingers could play notes 16 semitones apart, my index and pinky fingers could only stretch to play notes 11 semitones apart, which accounts for the gap found previously. To circumvent this problem, the user can create a text file that specifies their own finger constraints, which is passed as an argument and read at runtime. Each line of the text file is written in the following format:

```
'pinky', 'thumb', max_distance = 16, max_can_be_different_colors = False
```

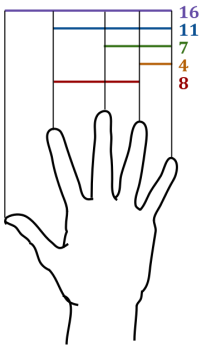

Figure 3 -
Example maximum finger distances used for testing, in semitones. Based on measurements of the author's hand.

When the program is executed, the text file is immediately converted into a list of lists. For each chord, the algorithm loops through the list of user constraints, referencing the previously created index list as indication for where each finger will be. The algorithm uses the following rules when looking at the constraints:

- Chords requiring 0 or 1 fingers are assumed to be playable.
- Chords requiring 2 fingers look only at the thumb and pinky constraint. This is assumed to be the largest specified interval, so if it is possible to play with any combination of fingers, it will also be possible with those two.
- Chords requiring 3 fingers assume the thumb and pinky play the exterior notes (same reason as above) and will test all three possible middle fingers for the remaining note before claiming it is impossible.
- Chords requiring 4 fingers assume the thumb and pinky play exterior notes (same reason as above) and will test all three combinations of two middle fingers for the remaining two notes before claiming it is impossible.
- Chords requiring 5 fingers check every constraint.

When an impossible chord is found, the split point must be moved. With these two constraints in place, each individual chord could be correctly validated as playable or unplayable based on each user's hand limitations.

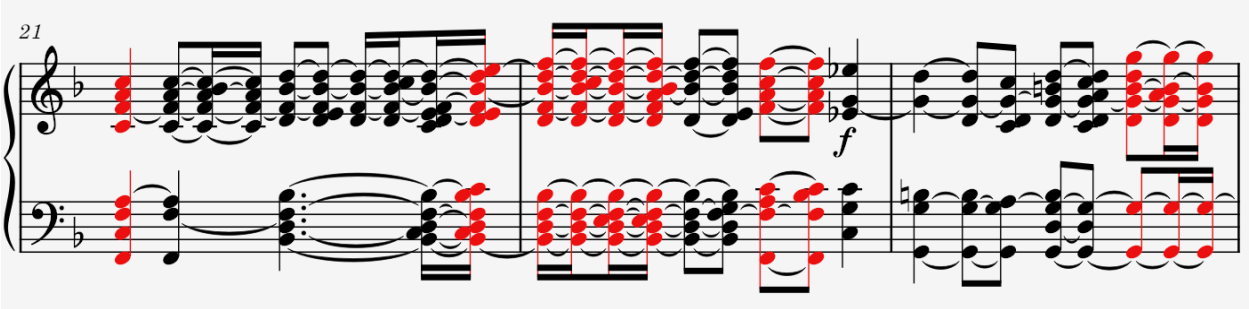

Figure 4 -
Excerpt from Sanctus a 12 by Giovanni Gabrieli. Black chords are possible, red chords are impossible.

Tied Notes

The final constraint that limits playability by a single pianist are limitations related to tied notes. In sheet music, notes that are tied are meant to be held and played as a single note. In the figure below, some notes are connected by curved lines.

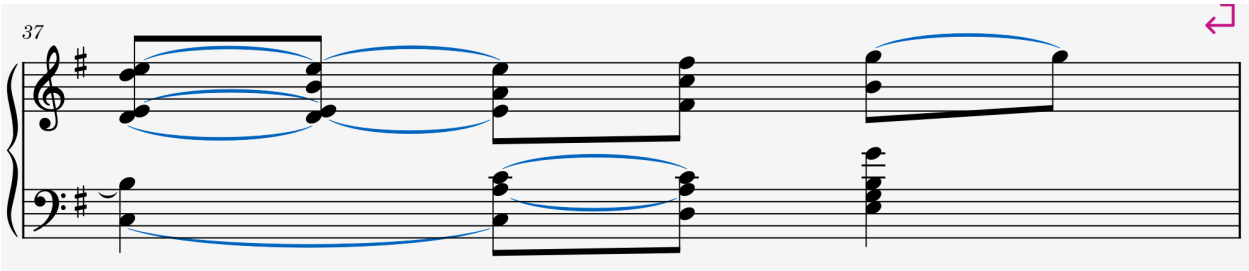

Figure 5 -
Excerpt from Missa Brevis in G by Mozart. Ties are highlighted in blue.

These lines are called ties, and they indicate that the notes they are connected to should be held, while the other notes are played normally. This essentially locks certain fingers in place. However, if not specifically addressed, the algorithm can split tied notes between hands, which should not happen. This is illustrated in the figure below, where the colored notes should be tied together.

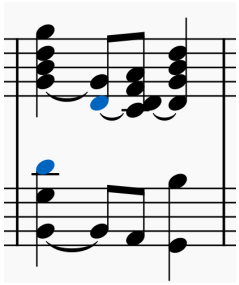

Figure 6 -
Notes colored blue are tied in the original score, but have been split between two parts, breaking the tie.

To address this problem, an additional constraint is placed on each left- and right-hand chord: if a note of the chord is tied to a note in the opposite hand's previous chord, it is deemed impossible and the split point must be moved. This, like with the previous constraints, occurs until both

hands' parts are playable or the chord as a whole is deemed unplayable. Tied notes can create interesting new reasons for chords to be unplayable.

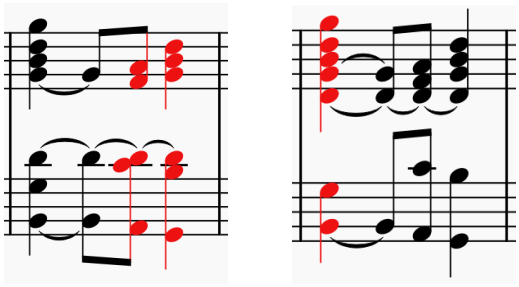

Figure 7 -
Left: Excerpt with tied note D4 in the left hand

Right: Same excerpt as left image but with tied note D4 in the right hand

The two excerpts above contain identical notes, with the only difference between them being the tied note D4 being played in different hands. When the tied note is played in the left-hand, the first chord is playable, but the third and fourth chords are not. When the tied note is played in the right-hand, the opposite becomes true. Even though each chord is entirely playable individually, the passage as a whole is always unplayable because the tied note causing one finger to be locked in place prevents essential movement of the hands and fingers. This is not always the case however, so if a tied note ever causes a chord to be unplayable, the program must check to see if playing the note with the other hand makes it playable again.

The algorithm that switches a tied note from one hand to the other is called only if a tied issue is present. It first looks at the current chord and moves the tied note to the other hand's chord. It also moves any additional notes that should be moved; moving a tied note from the left-hand to the right-hand should also move any notes above it, and moving it from right to left should move all notes below. Once complete, the algorithm traverses backwards through each chord, repeating this process until the start of the tie is found. As the chords are edited, their new

contents are tested for playability by the other two constraints. Once the algorithm has finished, if the passage is still impossible, the piece as a whole is considered unplayable. Additionally, if the new location of the tied note causes a greater number of impossible chords than there were originally, the tied note is then switched back for legibility of the output score.

Post Processing

Once the program is done validating the score using the three constraints, the boolean success value is known–the piece is either playable or unplayable by a single pianist. The final step is to create sheet music for a pianist to utilize. A new score object is created alongside a StaffGroup for the left- and right-hand parts. The piano staff is inserted, and the left- and right-hand parts that were constructed throughout are appended. Afterwards, the final score is written to a file. Without any additions, the output file would look like the figure below.

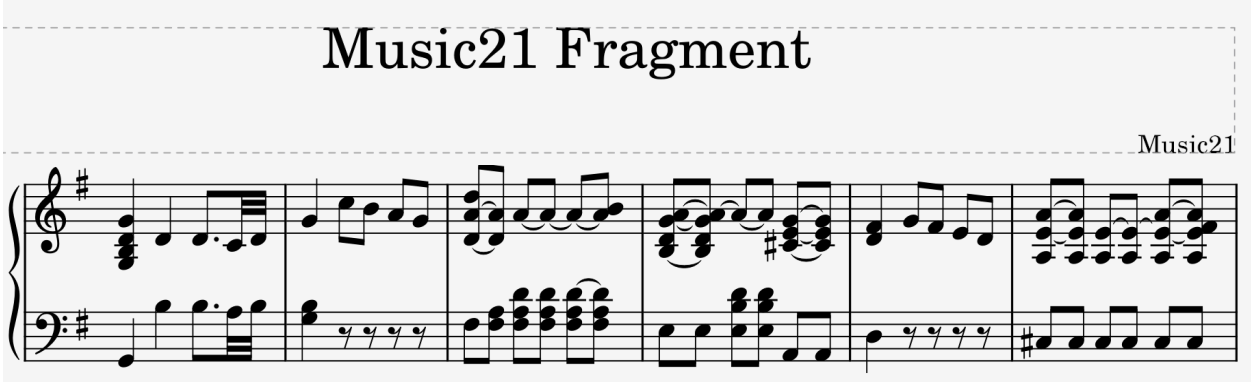

Figure 8 -
Raw output of the algorithm, containing repeated tied notes in measure three, repeated eighth rests in measures two and five, and missing title, composer, and dynamics.

This score is confusing to read and missing vital information. The first problem to be addressed was the repeating tied notes and rests. An unfortunate side effect of the chordify() function is that for every new note in any part, an entirely new chord must be created to accommodate where that note is and is not played, connecting the other notes together with ties. This causes

some instances where after the single part is broken up into two parts, there are tied chords and rests that are exactly the same and should be combined, as shown above. In order to fix this legibility issue, the program takes each completed part and traverses through each measure. For each measure, the program looks at each chord and rest from back to front, and if two adjacent objects contain the same harmonic contents and are tied together, they are combined. This is achieved by removing the current object and adding its duration to the previous object's duration.

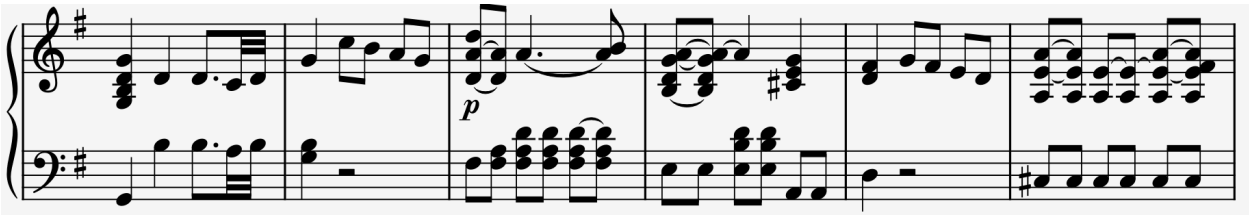

Figure 9 -
Same passage as Figure 8, but repeated tied notes and rests have been merged.

The next problem to be solved is the missing or incorrect information, such as the instruments, dynamics, and tempo. To fix the instrument problem, all existing instruments from the original score must be deleted, so the program removes each using a Python for loop. Afterwards, a Piano instrument is inserted to both hands' parts at offset 0, then manually named 'Pno'. To fix dynamics, the algorithm searches for the dynamics in the first part of the original score, which are then inserted into the new piano part at the correct locations. This is not a perfect solution, but it generally works well and is within the scope of the project. Tempo, song title, composer, and more, are all stored in the metadata of the original score, so when creating the new file, I insert metadata into the new file and set it equal to the metadata of the original. To finish post processing, I also remove the automatically inserted treble from the left hand part and insert a bass clef at the beginning instead. This change is not essential, but is an improvement in terms of convention and general legibility.

Merits

The Automated Music Transcription project aims to create a program that can take any existing musical score and produce the best possible arrangement using a set of specified instruments that maintains the integrity of the original piece while ensuring each part is playable by its assigned instrument. This algorithm would be of great use to musicians looking to create a cover of an existing piece if they aim to use a different set of instruments than the original piece was written for. There are many examples where this would benefit musicians. Three such examples include: cover bands who have a limited set of instruments/members, musicians looking to play scores that were not written to be played live (such as video game soundtracks, often containing many synthesized elements), and composers who want to ensure every part of the score they wrote is playable by the instruments they were written for. Software such as MuseScore allows musicians to quickly arrange parts for different instruments, allowing users to select a part and change the instrument it is played by. However, no software exists to validate the playability of these parts for the instruments they were designed for, nor is there software to automatically arrange scores for a different set of instruments–especially none that look at feasibility and creating an arrangement that makes the most musical sense. This research project aims to fill a gap in the music world that has surprisingly never been filled. Right now, musicians who look to rearrange existing scores typically do so manually, which can be extremely tedious and time-consuming. It is also prone to errors if the musician doing so has limited knowledge of what makes a part playable or not for every instrument. Additionally, they may remove notes or even entire parts they think are unplayable with their set of instruments, but in reality these notes and parts may be able to be maintained if they are arranged differently. The Automated Music Transcription project aims to fix all of these problems, making the

rearrangement of existing scores a painless process and allowing musicians to focus on learning and playing their music instead.

The piano validation algorithm marks a significant jump in progress towards the overall goal of this research project. No previous contributions have created algorithms to validate the playability of polyphonic instruments in any capacity–they have all been limited to monophonic instruments. Results show that the algorithm successfully enforces finger use limits, tied-note constraints, and realistic hand and finger spans based on each user's own specified restrictions. The program yielded 100% accuracy on all test cases, with both playable and unplayable chords being correctly identified using the restrictions specified by the user, thus yielding 100% accuracy on determining whether a score is playable overall by a single pianist. With these positive results, this project demonstrates that a constraint-based system can reliably validate musical parts for polyphonic instruments by determining the physical playability of each chord in the context of the piece. These findings will be useful to the future student researchers who work on this project as it continues to progress towards its final state.

In isolation, the piano validation algorithm is still useful to pianists and composers alike. Any pianist looking to perform a piece not originally written for piano can easily convert any score to piano sheet music. While no reductions are ever made by the algorithm, impossible chords are very visible in the output score. In these instances, pianists can choose which notes are most comfortable to play and drop those that are not. This saves an immense amount of time from transcribing the notes of every part of a score by hand. Composers who do not play the piano can also benefit from this algorithm, as they can verify if what they wrote can actually be

played by a professional. If not, they can easily see which chords need to be adjusted and change them accordingly.

Challenges

Some aspects of the Music21 library are very intuitive, but many aspects are not. The biggest challenge by far was keeping the rhythm of .musicxml output the same as the input score, and this is for three main reasons. When inserting a note into a part, you must specify two things: the measure, and the offset. A measure is a subsection of a piece, where each measure is contained within two vertical lines in the staff. In Music21, measures are objects, so to insert a note you must insert it into the correct measure object. The other thing you need is an offset, or the number of beats into the measure the note should be placed. For example, inserting a note at offset 0 inserts it at the very beginning of that measure, or the first beat. The challenge arises when realising that offset values are only reliable while reading values, not while writing. During insertion, the offset attribute acts only as the order in which notes in a measure should be arranged; it does not correspond to actual offset values. For example, a note at offset 0 will be correctly inserted before a note with offset 1, but the note with offset 1 will only be placed at the second beat if the first note is a quarter note (a note with a duration of one beat). Music21 assumes that any inserted notes already line up the other notes, so they do nothing to fix user error of this kind.

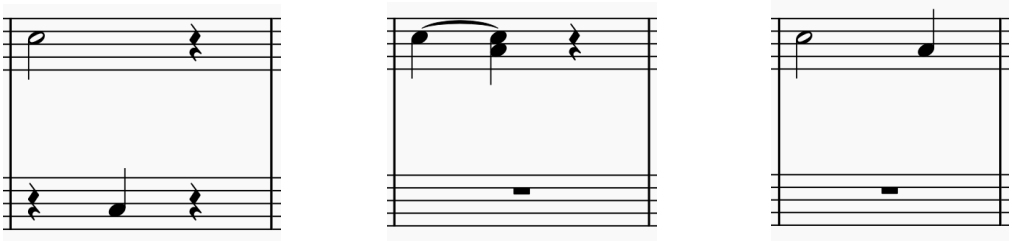

Figure 10 -

Left: Input score. The top note starts at offset = 0.0 and plays for 2 beats. The bottom note starts at offset 1.0 and plays for 1 beat. This means that the bottom note plays on the top note's second beat.

Middle: Ideal output when inserting both notes into a single part: the bottom note starts at offset 1.0. This is what happens when you use the chordify() function.

Right: What happens when the bottom note is inserted at offset = 1.0. It is placed at offset = 2.0 because of the top note's duration. When inserting, offset only dictates the order notes appear.

This was a challenge for the first step of this project, which was combining every part of a score into a single part, as oftentimes the notes of different parts do not line up. The naive approach of inserting each note from each part at their original offset completely changes the rhythm of the output unless the rhythm of every part is exactly the same. Luckily, I did not have to deal with this problem on my own, as the chordify() function solves it perfectly. Whenever a rhythmic problem arises, chordify() splits the notes into multiple tied notes. This results in an output where a new chord is created anytime a note is played in any of the original parts.

The second rhythmic challenge is an extension of the first, but one that I needed to solve myself. When splitting chords between the left- and right-hand parts, sometimes a hand is left with a rest, meaning that it plays no notes for a certain duration. Thanks to the offset attribute of insertion only acting as a sorted order, any rests that should exist are erased, with phantom rests" being automatically inserted at the end of the measure after all the notes have been played.

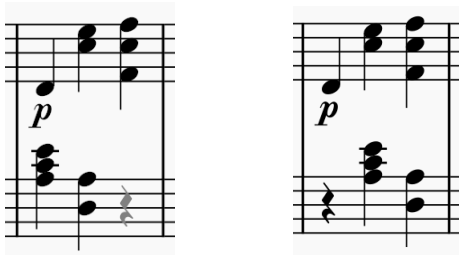

Figure 11 -
Left: Chords in the left-hand part inserted at offset = 1.0 and offset = 2.0

Right: Corrected version with a rest inserted at offset = 0.0

In order to solve this problem, the program must find where every rest should be located and insert them properly. This is first done by finding points in the score that no notes should be played at all, in any part. These rests are inserted into both the left- and right-hand parts at the correct offset and for the correct duration. The rest of the algorithm then executes as normal. Once it has been completed for a specific chord, the program checks to see if one of the hands' parts is empty. If so, a new rest object is created and inserted into the measure in place of an empty chord, which does nothing.

The final challenge with maintaining rhythm is the hidden connection between notes and their chords in Music21. As I have mentioned previously, after the validation algorithm is complete, tied chords that can be combined in a part are combined for legibility. This works by removing a tied chord and extending its predecessor by its duration. An unexpected consequence of this is that the chord played by the other hand at that offset is also extended, even though only one hand is passed as a parameter at a time. I learned that this is caused by notes being intrinsically connected to chords. Even though the chords of each hand are different chord objects, the notes that make them up are the same note objects from the original chord of the combined score. This means that if one of them gets extended, they all get extended.

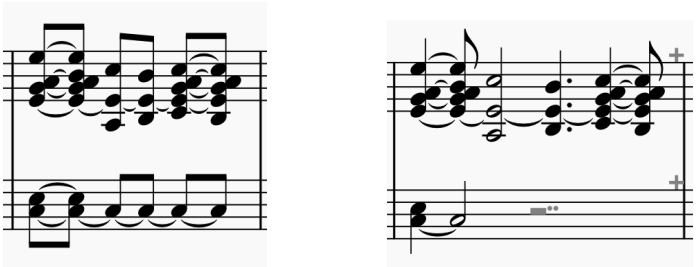

Figure 12 -
Left: Excerpt with correct rhythm, but equal tied chords have not yet been connected.

Right: The 4 tied eighth notes in the left-hand are correctly converted to a half note, but each chord in the right hand starting from the end incorrectly increases their duration by an additional eighth note each chord as the left hand notes were being merged.

As seen above, this behavior adds additional beats to one of the hands, which in turn creates more phantom rests in the first hands' measure to keep the number of beats the same. I solved this problem by deepcopying the two lists containing the left- and right-hand notes before creating new chord objects with them. This breaks the notes' connection with their original chord as they are now different objects than the notes contained within the original chord object. Now, extending one of these notes has no effect on the others, solving the final challenge of maintaining the original rhythm.

Limitations

The algorithm determines physical playability of each chord with overwhelming success. However, there are some limitations that prevent the program from being completely accurate in other aspects. The first is that tempo is not taken into account when looking at the playability of a piece. If a piece of music is too fast for a human to play, or the pianist has to jump around the keyboard faster than realistically possible, the program may say the piece is still playable as long as each chord is physically possible for a human hand to produce. When considering speed as an obstacle for playability, it is extremely difficult to determine which pieces are too fast for a human to play versus which pieces take additional skill and practice to reproduce. For this reason, Dr. Papamichail explicitly stated that tempo and speed should not be taken into account in this particular algorithm when trying to determine playability. Unfortunately, this means that some extremely fast scores may make it through the algorithm without getting marked as impossible for a single pianist, even if the speed of the notes do in fact make it humanly impossible.

Another limitation of the program comes from overlapping notes. One of the primary goals of this research project is to preserve the integrity of the original piece as much as possible, and there is one instance where my program falls short in this regard, and that is when two separate parts play the same note at the same time. When the chordify() function is used and the notes of each part are combined into a single part, any overlapping notes must be condensed in some way due to limitations of music notation and sheet music in general. Even if sheet music could account for this, the physical limitations of a piano would also not allow for these cases to exist, no matter how many pianists you had on a single piano. There are two cases where this limitation occurs. The first case is if two instruments play the same note at the same time for the same duration. This is the better of the two cases, as the only thing that is lost is emphasis of said note. In the original score, two instruments are playing the same note, but in the piano rendition, the note can only be played one time. The note is still being played at the correct time and for the correct duration, so very little is lost in terms of integrity. The second case is if one part plays a note for a long duration while the other part plays the same note for a shorter duration.

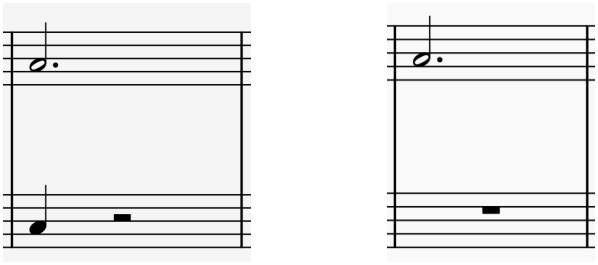

Figure 13 -
Left: Two parts of an input score that play the same note for different durations.

Right: Left- and right-hand parts after going through the algorithm. The shorter note has been erased entirely.

In this case, the chordify() function must pick a note to prioritize. Either the longer note must be shortened or stopped to account for the melody note, or the melody note must be lost in favor of

the sustained note. Either way, noticeable integrity is lost from the original piece, as a piano cannot play both notes at once and separately at the same time. In our research meetings, Dr. Papamichail stated that this problem should be ignored, as realistically there is no solution; integrity will always be lost in these instances if the piece is to be played on a single piano. When this algorithm is combined with the remainder of the research project, another instrument's part would be allowed to take an overlapping note, solving the issue of integrity.

The final limitation of this project is that while some chords may be physically possible for the hand to play, some chords in the output are not easy to play. Due to the nature of part combinations, chords will often span much of the piano or become extremely note-dense, leading to chords that are extremely difficult or uncomfortable to play, even if technically possible by the human hand.

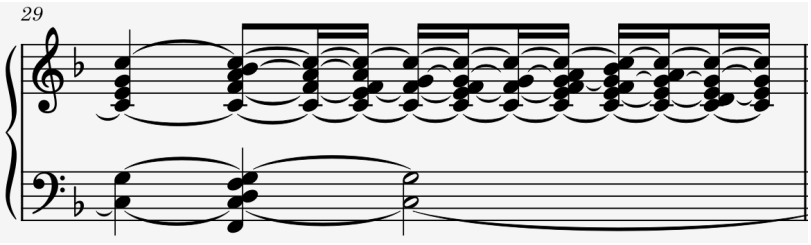

Figure 14 -
Excerpt from Sanctus a 12 that is playable but extremely note dense.

Because these combined chords were not written with a pianist in mind, the large chords produced can be strenuous for players' fingers, oftentimes requiring them to stretch to their limit. The algorithm tries to find any way to arrange the left- and right-hand parts to be playable for the user, leading to many unusual sequences and movements in scores with many parts.

Future Work

The future of the Automated Music Transcription project as it relates to my contributions can be broken down into three parts. The first are additions to the piano validation algorithm itself by adding a set of competencies to the validations. This means that in addition to users being able to specify their own hand constraints, they would also be able to specify piano techniques and mastery levels that they possess or do not possess. This would improve the program by determining whether a piece is actually playable by a specific user rather than testing if each chord is physically possible for their hands to play. Some competencies could include a maximum speed of notes and chords, or a maximum number of notes per hand. These additions would solve two of the previously mentioned limitations, being tempo and uncomfortable chords, which would strengthen the quality of the algorithm's output significantly.

In addition to improving the piano validation algorithm, future research will involve expanding these validations to other polyphonic instruments, such as guitar. Each polyphonic instrument has its own unique set of constraints, so what is playable on one may not be playable on another. However, using my project as a foundation will give future students an easier time understanding how to look for such constraints and how to implement them for their own instrument validation algorithms. Eventually, student researchers will likely create a general polyphonic validation framework with customizable rules for each instrument. This may involve a large list of polyphonic constraints broken up into separate functions that are either called or not called depending on the instrument being validated. Extending the functionality of my algorithm to other instruments would be a significant leap in progress for the polyphonic phase of the research project, which has only just begun with my contribution.

In the long term, the future work of this project will involve integrating my piano validation algorithm, as well as any future polyphonic validation algorithms, into the research that has already been done in pattern recognition, reduction, separation, and arrangement. Once the polyphonic validation system is complete, student researchers will be responsible for using every previous students' algorithms to create a true automated music transcription algorithm. This solves the final unresolved limitation of my algorithm, that being loss of integrity due to overlapping notes. Scores will be broken down into their parts, separated and combined into new parts, and reduced for playability for a completely new arrangement of instruments in the most faithful way possible. The result will be a complete system capable of taking any score and arranging it for any combination of monophonic or polyphonic instruments–an invaluable tool for musicians everywhere.