Chapter 5

Results and Future Work

This chapter will describe the results of my work using an excerpt of of W. A. Mozart's *Eine kleine Nachtmusik, K.525- I.Allegro* (referred to by just "K.525" from here onwards) as an example. It will also use an excerpt of W. A. Mozart's *String Quartet No.7 in E-flat major, K.160/159a- I.Allegro* (referred to by just "K.160" from here onwards)as an example where K.525 doesn't fit the specifications. This chapter will also look at timing metrics of the system.

5.1 Results: Example 1: Eine kleine Nachtmusik, K.525- I.Allegro, W. A. Mozart)

' This section will go through an example of how the entire `OMRMIDICorrector` system works on W. A. Mozart's *Eine kleine Nachtmusik, K.525- I.Allegro* .

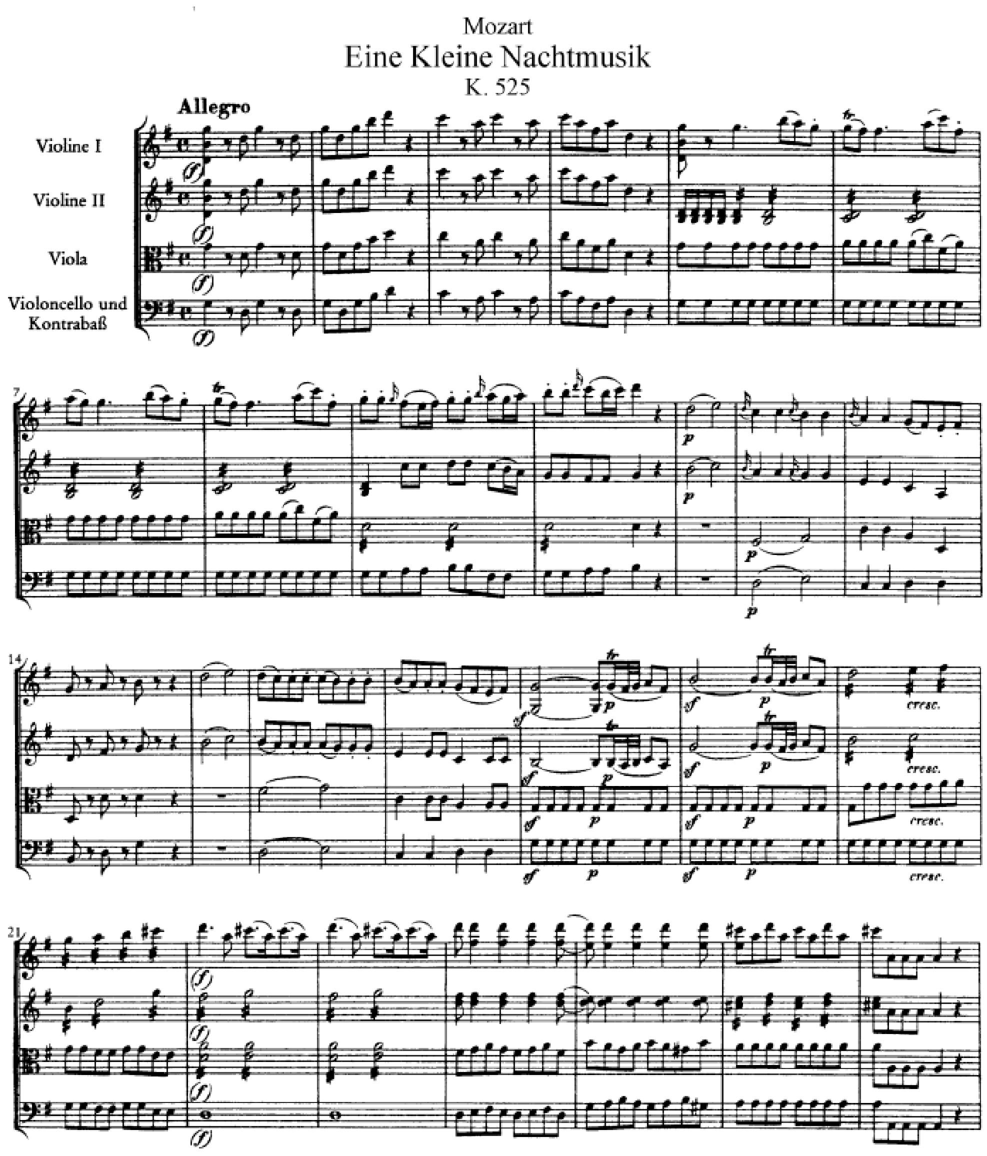

Figure 5-1: The first page of the scanned copy of the score found on IMSLP.

5.1.1 Musical Properties of K.525

I chose K.525 as the example piece for many of it music properties that make it a difficult but interesting piece to align.

5.1.1.1 Bass and Cello Doubling

In the original scanned score, there are only four parts because the bass part doubles the cello part one octave lower. However, in the MIDI encoding, there must be five separate voices. Thus, in the preprocessing step of the `OMRMiDICorrector`, it would be ideal for the system to recognize this. Otherwise, in instances where the `OMRMIDICorrector` could not pair up the parts between the input OMR stream and the input MIDI stream, there would be an error thrown, and alignment and correction would not happen.

5.1.1.2 Tremolos

Starting in measure 5 in the original scanned score, the second violin has tremolo notes, instead of repeated 16th notes written out. The OMR parser in SmartScore is not robust enough to recognize the slashes across the stem of the note as tremolo markings and instead interprets it as a single note. The MIDI protocol has no way of indicating tremolos, so the second violin's notes in measure 5 are encoded as regular 16th notes. Ideally, the aligner would be robust enough to align tremolo measures with non-tremolo measures even though there will be a huge discrepancy in the number of notes present in the measure.

5.1.2 Preparing the Raw Input

The basic raw input of the system is an OMR score and a MIDI file. A scanned copy of the score was found in IMSLP [6], the source for much free public domain sheet music. The MIDI file was sourced from the Yale MIDI Archive.

I used SmartScore's built-in OMR tool to convert the scanned score of K.525 into an ENF file. Then I exported the post-OMR file as a musicXML file.

Similarly for the K.525 MIDI file, I used SmartScore to parse the original file and exported it as a musicXML file.

The last step of preparing the raw input is to parsing the musicXML files with the `music21` library so that they are `Stream` objects. This can be done by passing in the musicXML filepaths to the `parse` method in the `converter` class. We will call these two parsed streams `k525midiStream` and `k525omrStream`.

```
from music21 import *
k525MidiFilepath = "pathto/k525mvmt1/miditoxml.xml"
k525OmrFilepath= "pathto/k525mvmt1/omrtoxml.xml"
k525midistream = converter.parse(k525MidiFilepath)
k525omrstream = converter.parse(k525OmrFilepath)
```

5.1.3 Running `OMRMIDICorrector` on K.525

After the raw input has been massaged into `Stream` objects, the aligning process is simple. We create an instance of an `OMRMIDICorrector` object with the two streams as parameters. For the purposes of this example, we will use the default hasher that is built into the `OMRMIDICorrector` class. We will set the `debugShow` to `True` so that we can visualize the alignment between pairwise streams. Finally, we will call `processRunner` which will hash and align the parts in the stream. For now, we will manually create the `Fixer` objects for stream fixing. The aligner is able to correctly identify that the repeated 16th notes

```
from music21 import *
OMC = alpha.analysis.omrMidiCorrector
k525omc = OMC.OMRMIDICorrector(k525midistream, k525omrstream)
k525omc.debugShow = True
k525omc.processRunner()
```

These are the outputs of the `showChanges` function.

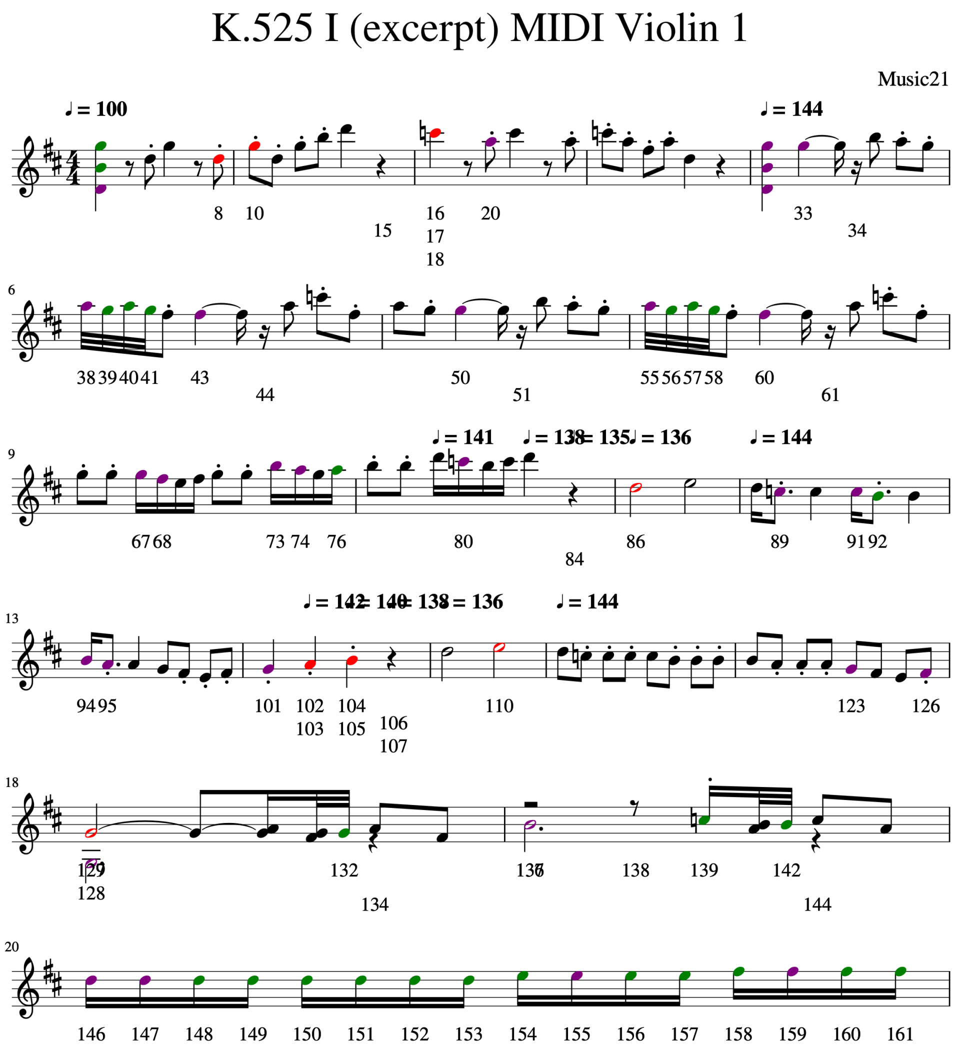

Figure 5-2: Post-alignment visualization of an excerpt of the MIDI Violin 1 part in K.525- I.Allegro.

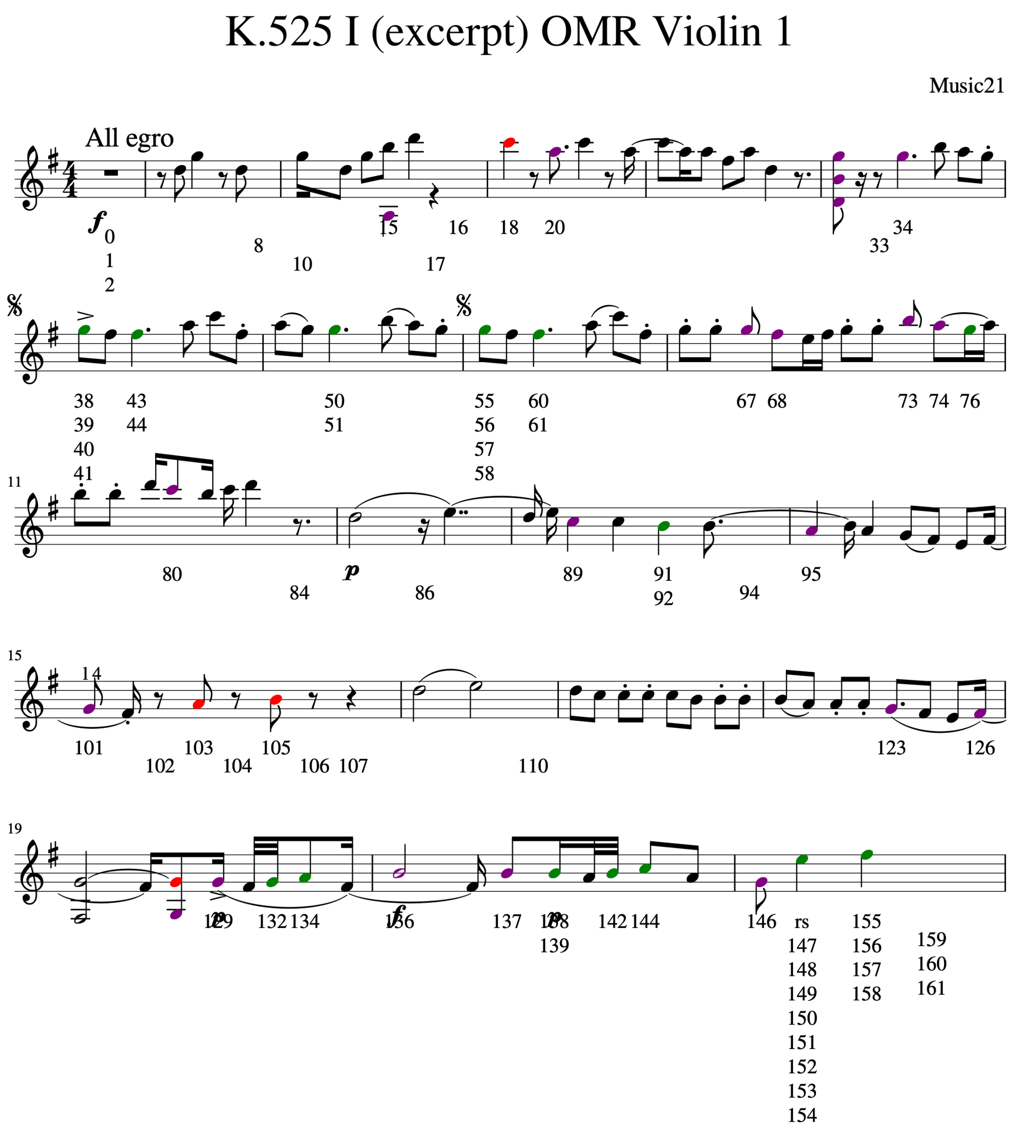

Figure 5-3: Post-alignment visualization of an excerpt of the OMR Violin 1 part in K.525- I.Allegro.

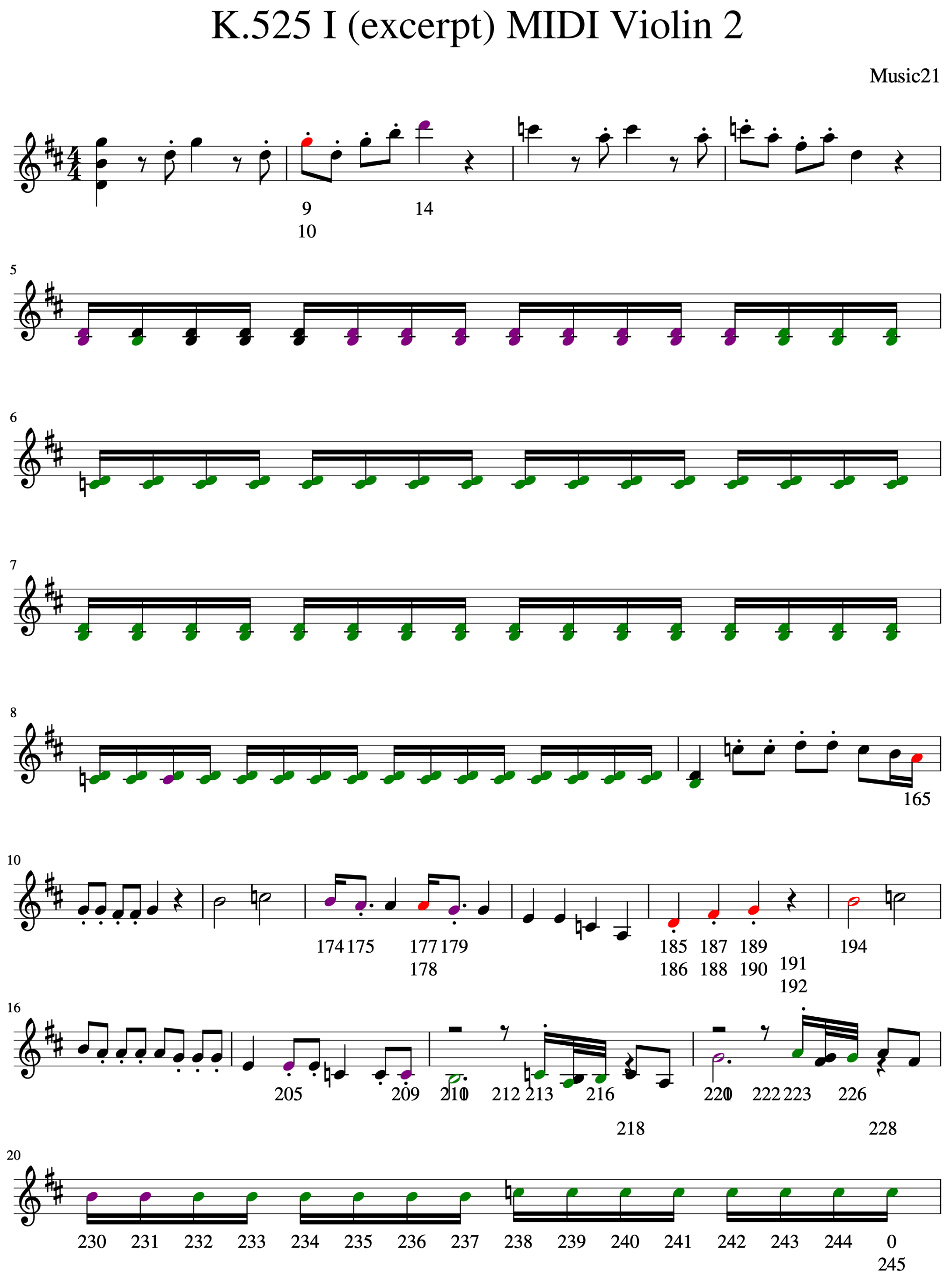

Figure 5-4: Post-alignment visualization of an excerpt of the MIDI Violin 2 part in K.525- I.Allegro.

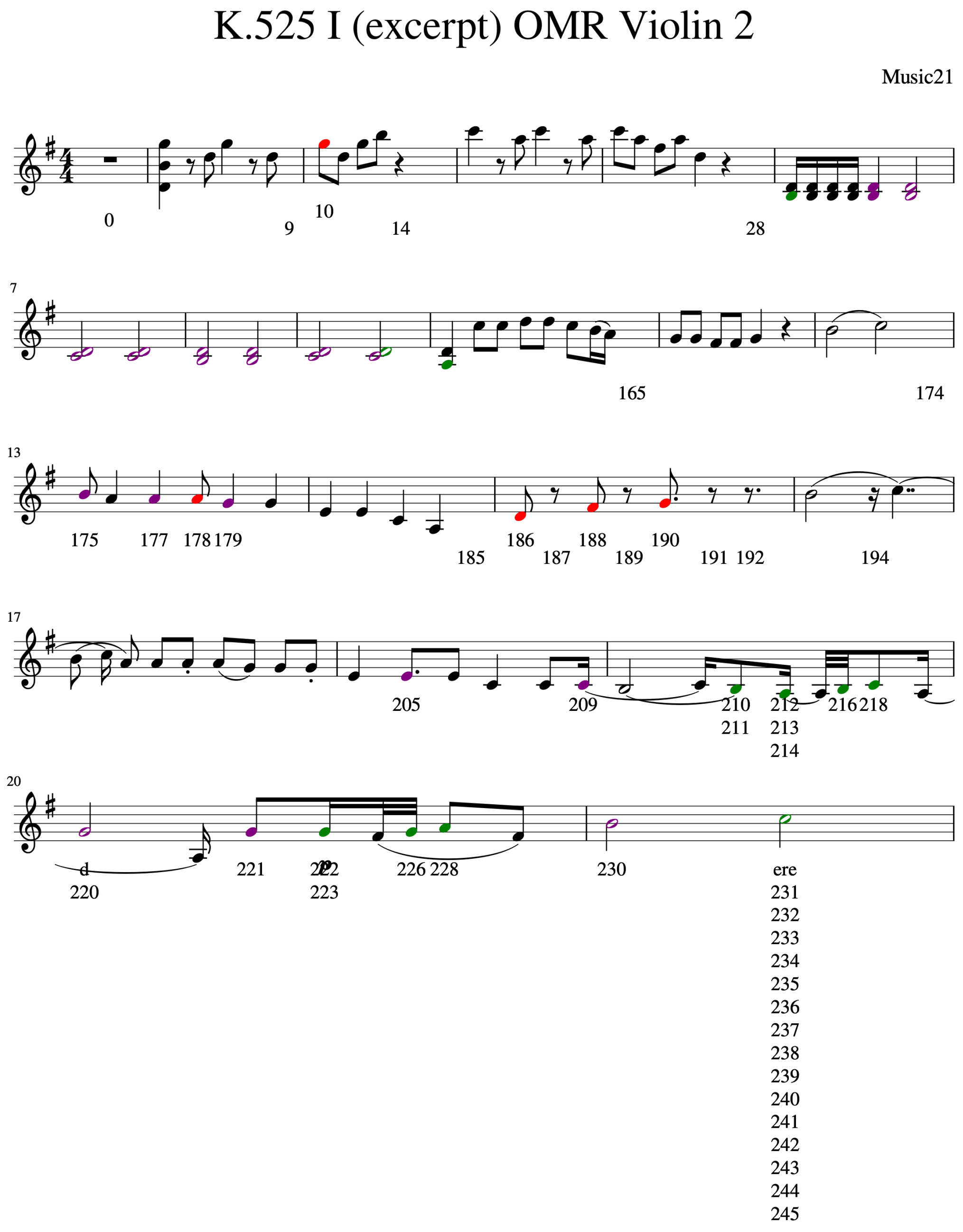

Figure 5-5: Post-alignment visualization of an excerpt of the OMR Violin 2 part in K.525- I.Allegro.

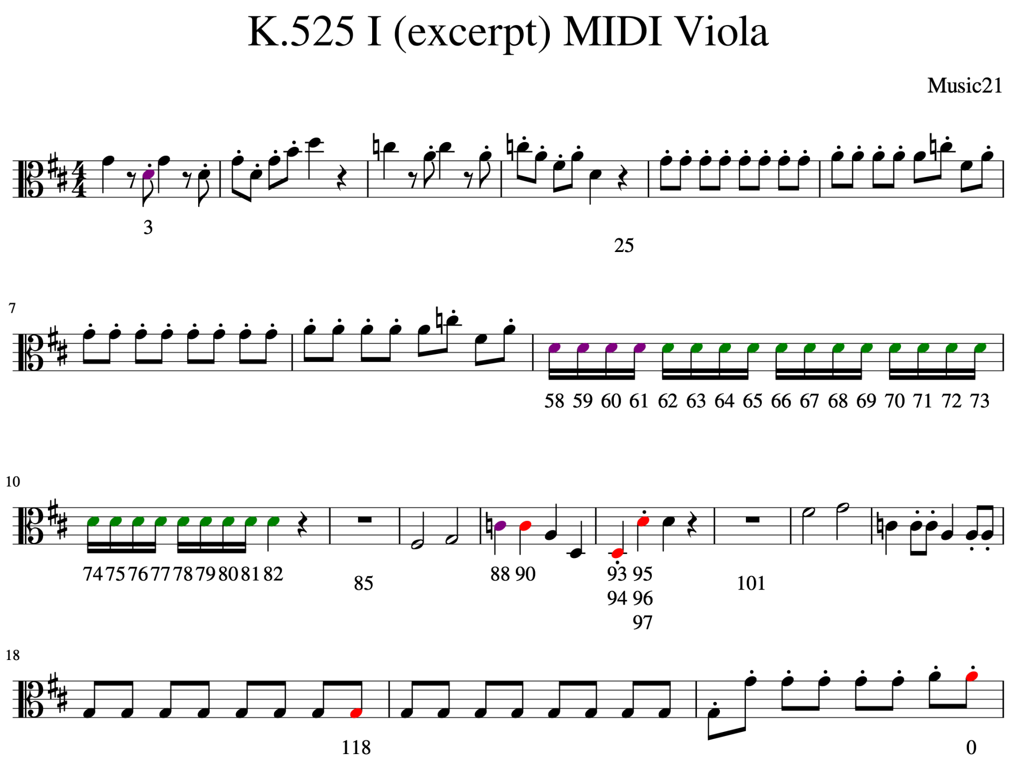

Figure 5-6: Post-alignment visualization of an excerpt of the MIDI Viola part in K.525-I.Allegro.

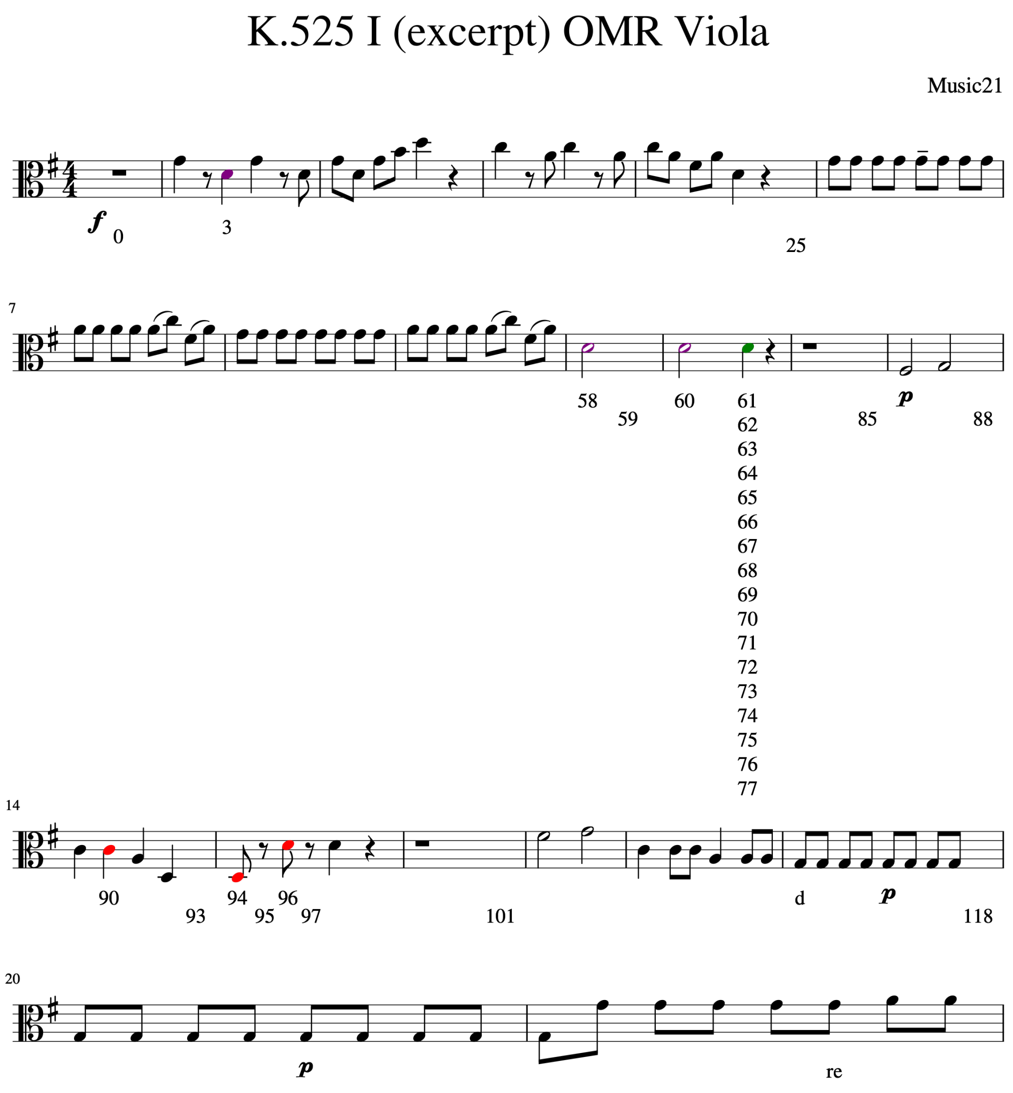

Figure 5-7: Post-alignment visualization of an excerpt of the OMR Viola part in K.525- I.Allegro.

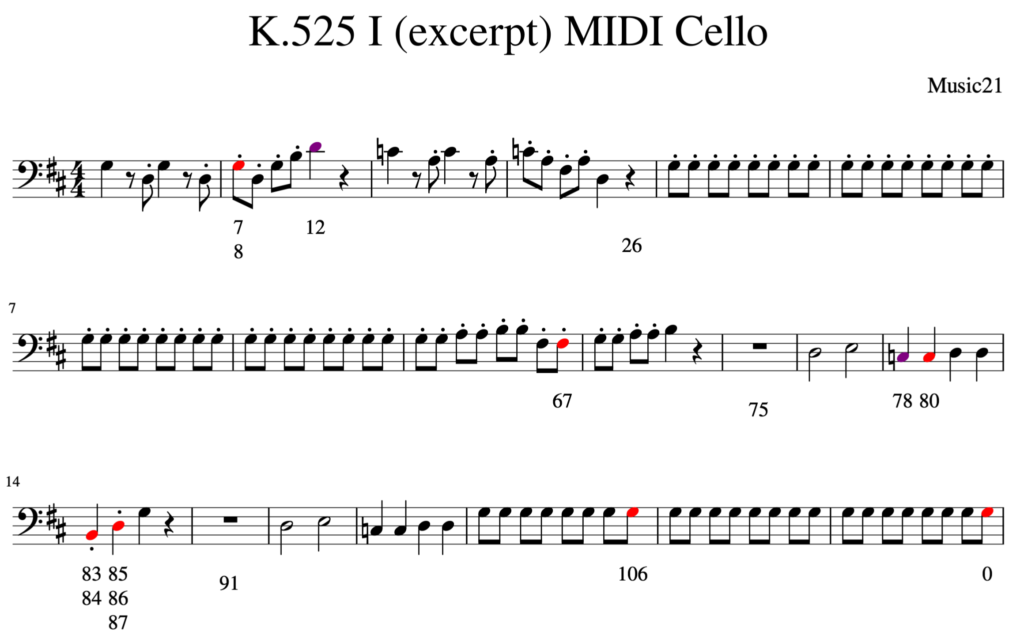

Figure 5-8: Post-alignment visualization of an excerpt of the MIDI Cello part in K.525-I.Allegro.

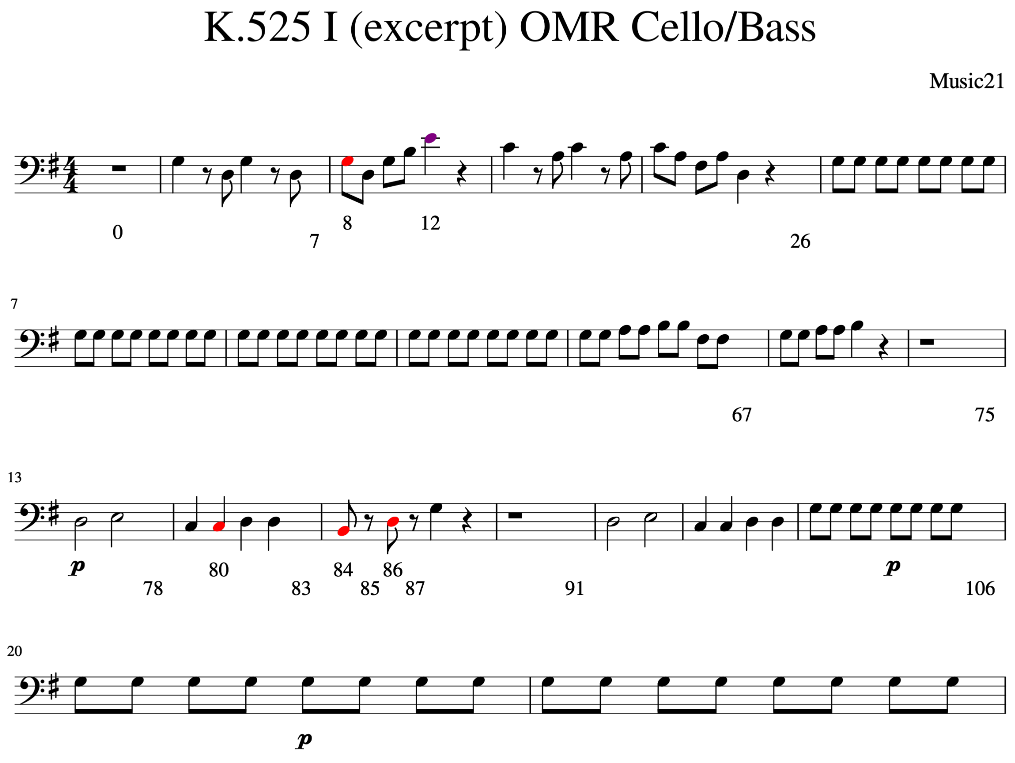

Figure 5-9: Post-alignment visualization of an excerpt of the OMR Cello and Bass part in K.525.

5.1.3.1 A Closer Look at Specifics of `processrunner`

A lot of magic happens within `processRunner`, all of which is described at a high level in sections 4.1.1.1 - 4.1.1.4. Here are some salient details of the process in the context of K.525 that I think deserve notice:

- Bass Doubles Cello Part - The OMR score has only four parts, whereas the MIDI score has 5. The `checkBassDoublesCello` method in `OMRMIDICorrector` is able to correctly identify this and thus does not reject the input on the basis of having a mismatching number of parts in the stream.

- Rhythmic Fault Tolerance: Beginning Rests - The first measure in the OMR score is a scanning and parsing error. The `Aligner` is robust enough to identify the beginning rest as a insertion error.

- Rhythmic Fault Tolerance: Propagating Rhythmic Shifts - By measure 5 in the Violin 2 part, we see that the OMR score has introduced a 16^{th} note shift in the rhythm which continues for a few measures.

- Notation Fault Tolerance: Tremolos - By measure 5 of the Violin 2 part and by measure 10 of the Viola part, the repeated 16^{th} notes get notated with tremolo slashes in the original OMR score. Of course, the MIDI protocol doesn't have a way of representing tremolo, so the notes are written out in long form. The corrector is able to match the written out repeated notes in the MIDI to the (incorrectly) scanned corresponding notes in the OMR.

5.1.4 Example 1: Similarity Metrics in K.525

Recall that the `similarityScore` is the ratio of `NoChange` operations to the total number of operations in the `changes` list. For each pair of streams in K.525, the `similarityScore` and `changesCount` are listed below:

	similarityScore	changesCount
Violin 1	0.51	{NoChange: 83, Ins: 30, Del: 10, Sub: 39}
Violin 2	0.30	{NoChange: 74, Ins: 125, Del: 9, Sub: 38}
Viola	0.72	{NoChange: 97, Ins: 21, Del: 9, Sub: 8}
Cello/Bass	0.87	{NoChange: 107, Del: 11, Sub: 5}

Table 5.1: `similarityScore` and `changesCount` for every pair of aligned streams

A visual examination of the pairs of streams suggests that these numbers and counts look correct. The Cello/Bass parts have the fewest colorations to their notes and no insertion changes at all. The least similar pair of parts is the Violin 2, which has multiple cases of the tremolo problem described above as well as propagating rhythmic shifts.

5.2 Example 2: String Quartet No.7 in E-flat major, K.160/159a- I.Allegro, W. A. Mozart)

This section will go through an example of how just the `EnharmonicFixer` works on correcting and excerpt of the Violin 1 part of K.160. The OMR score was sources from IMSLP [7].

5.2.1 Running `EnharmonicFixer` on K.160, Violin 1

After prepping the raw input files of K.160 in the same way as detailed above and running that through an `OMRMIDICorrector` instance, we will create an and `EnharmonicFixer` instance using just the Violin 1 OMR stream and MIDI stream and the associated `changes` list.

```
from music21 import *
k160MidiFilepath = "pathto/k160mvmt1/miditoxml.xml"
k160OmrFilepath = "pathto/k160mvmt1/omrtoxml.xml"

k160midiStream = converter.parse(k160MidiFilepath)
k160omrStream = converter.parse(k160OmrFilepath)
k160omc = OMRMIDICorrector(k160midiStream, k160omrStream)
k160omc.processRunner()

k160omcV1Changes = k160omc.changes[0]
k160omcV1Midi = k160omc.midiParts[0]
k160omcV1Omr = k160omc.omrParts[0]
k160EF = fixer.EnharmonicFixer(k160omcV1Changes, k160omcV1Midi, k160omcV1Omr)
k160EF.fix()

k160omcViolin1OmrStream.show()
```

Below, The first two figures below show the alignment visualization of the Violin 1 part. The last figure shows the corrected OMR score after it has been changed by the `EnharmonicFixer`

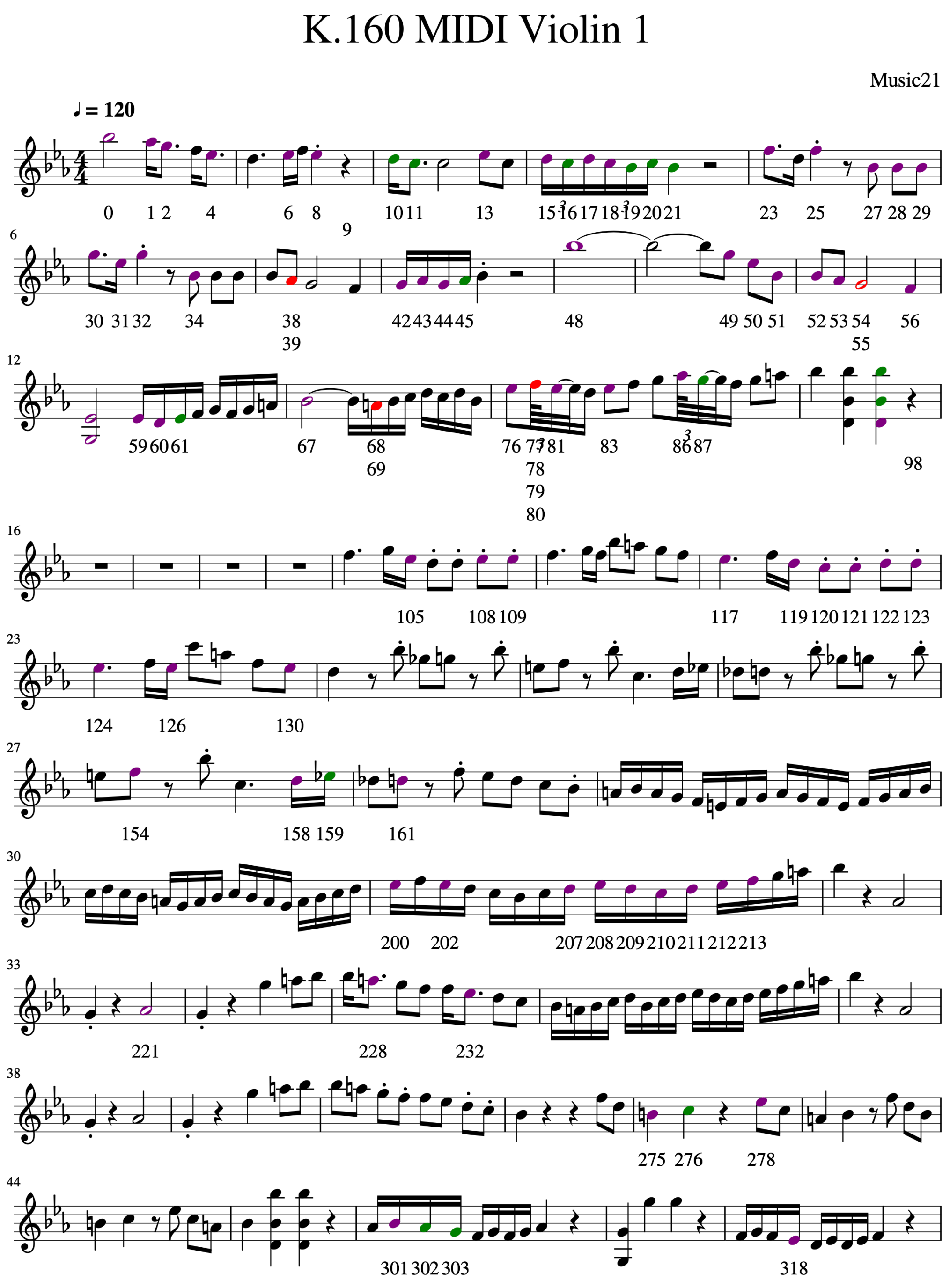

Figure 5-10: Post-alignment visualization of the MIDI Violin 1 part in K.160.

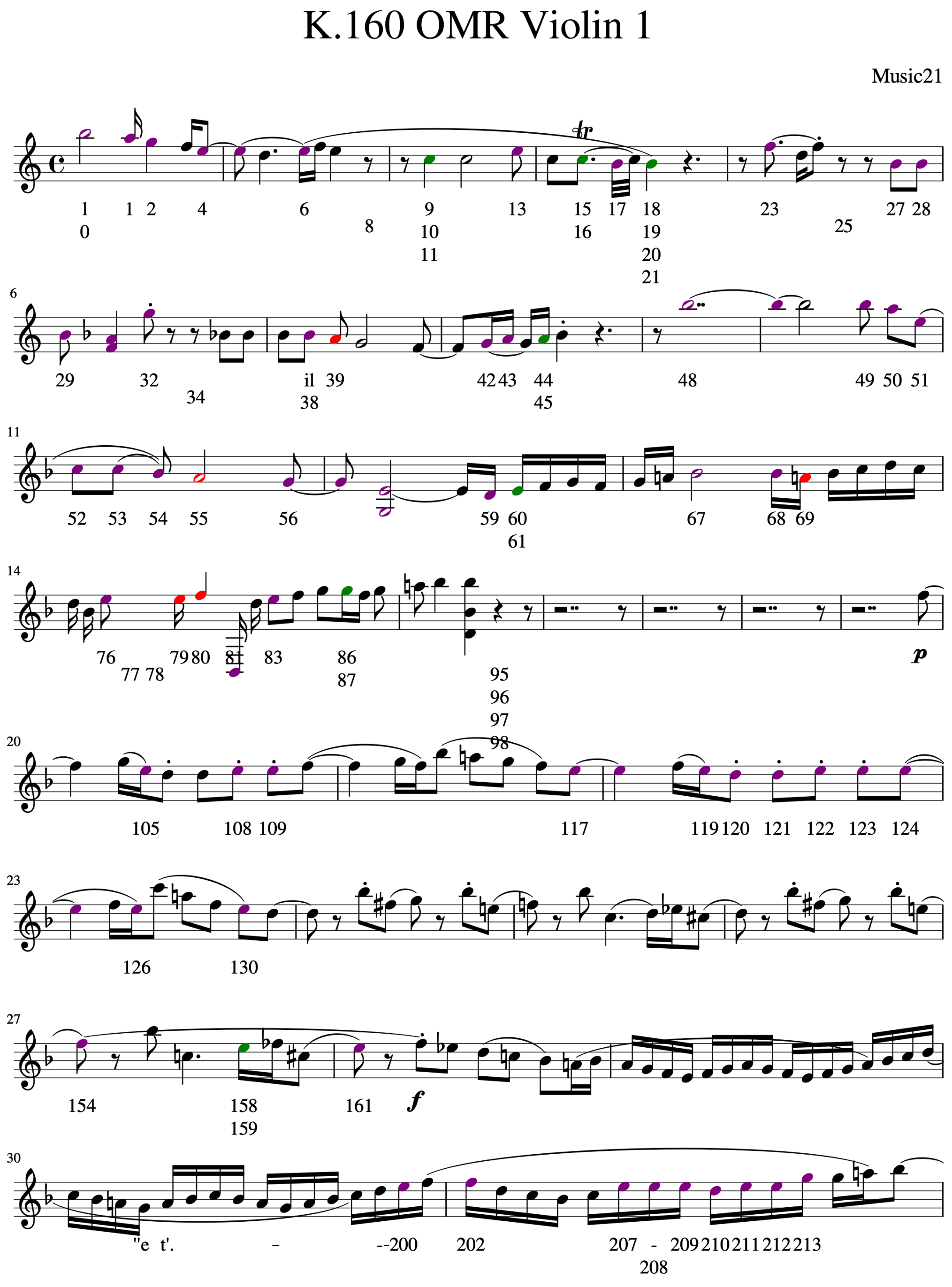

Figure 5-11: Post-alignment visualization of the OMR Violin 1 part in K.160.

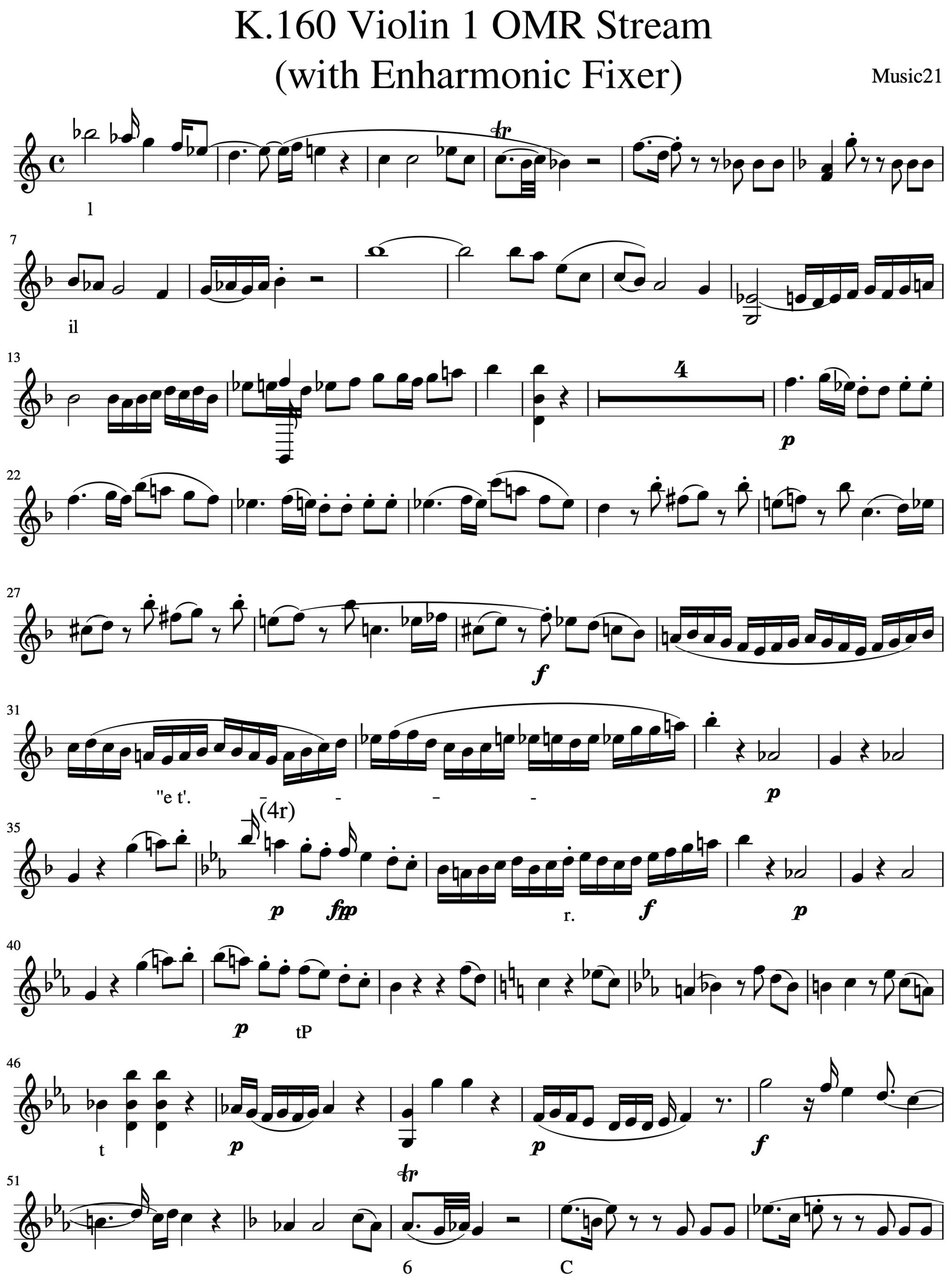

Figure 5-12: Post-enharmonic and pitch fixing visualization of the OMR Violin 1 part in K.160.

5.2.2 Results: Example 2: Analysis of Correction

Visual inspection of figures 5-10 and 5-11 suggests that the OMR failed to pick up the appropriate key signature of the movement and that is what lends to large number of changes in the first 10 or so measures. Inspection of figure 5-12 shows that many of the pitch errors have been corrected using the `EnharmonicFixer`. Of course, the most ideal scenario would be to recreate the key signature, but having the correct pitch is a good start.

5.3 Results: Timing Tests

This section will discuss and evaluate the system based on its runtime analysis and empirical results. It will also discuss how the system would scale to different sized inputs.

5.3.1 Runtime Analysis

There are three major modules of the system the `Hasher`, `Aligner`, and `Fixer`.

5.3.2 Empirical Results

5.3.3 Scalability

5.3.4 Implications of Free Music

The lasting and tangible results of this research will hopefully be in helping providing more free and high-quality musical scores to the public. With ISMLP alone having over 100,000 scores that are freely available, there is a lot of potential for my work to liberate otherwise sub-quality scanned musical scores and turn them into digital scores that can be conveniently analyzed, replicated, and distributed. It's my hope that this work will be able to facilitate musical research, provide individuals and groups with easy access to practice material, and help bring more music to all of our lives.