Four Part Voice Writing as a Constraint Satisfaction Problem

Andrew Crapitto, Sam Craig, Syed Quazi

January 23rd, 2022

Abstract

Four part voice writing is a concept in Western Classical music theory in which music is written for four distinct voices: soprano, alto, tenor, and bass. Throughout the common practice period, music written in this style developed distinct patterns and conventions to the point they could almost be described as rules. As such, four part voice writing is commonly taught to conservatory students in their music theory studies, with exercises given where the student gets some amount of information (e.g. one of the voices, and the harmony) and must write the rest of the music according to the conventions of part writing. To solve such exercises, we modeled four part writing as a Constraint Satisfaction Problem (CSP) and implemented a program in Python that finds all possible musical solutions that follow the conventions (constraints). The program takes as input and gives as output a file of .4pw format – which we designed – that has a one-to-one mapping to standard Western Classical music notation.

Introduction

As described above, four part voice writing is a very common exercise in the study of music theory. As such, there are a few motivations for writing an intelligent program to solve it. For one, having a program that can solve these exercises automatically would be useful for anyone tasked with completing a large quantity of them. We would never encourage cheating on an assignment of course, but the theoretical usefulness of the program is undeniable from that perspective. Furthermore, a computer has the ability to quickly and accurately find *all* possible solutions within the given constraints. And so it can be used by students or teachers of music theory to find more possible solutions than they may have been able to find on their own. Lastly, such a program could even be used – in theory – for novel music generation, which we will touch on a little more later.

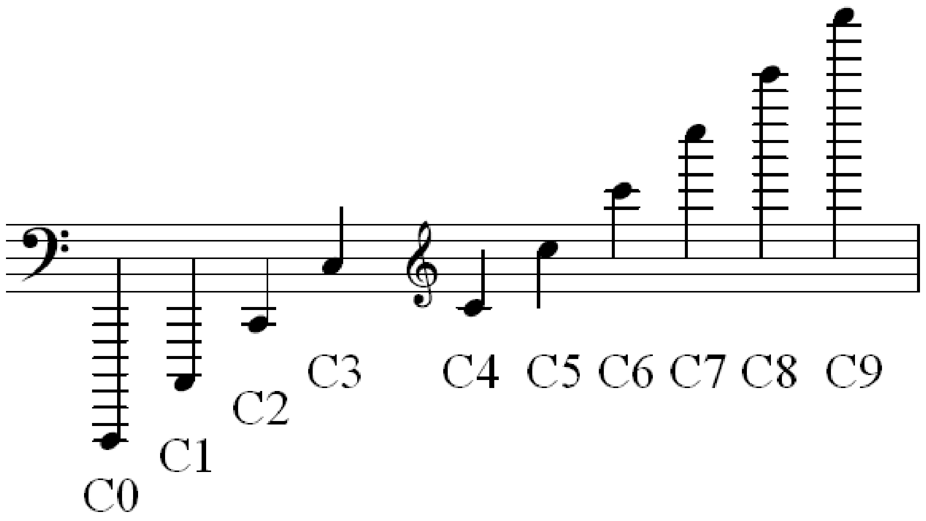
The specific problem being addressed by our project is the completion of music written in the style of common practice four part voice writing. That is, at a minimum our program must receive information about the harmony as input. The four voices can be filled in to any degree. Two voices could be filled and two could be unknown. All could be unknown. They could all be partially unfilled to some degree. The program simply needs the harmony, and it will solve what the pitches could be.

Our solution for this was to model four part voice writing as a CSP. In this model, the conventions that define the traditional relationships between different voices and the harmony are modeled as constraints on what the possible value for unknown pitches are. We created a mapping from standard musical notation to a mathematical representation of pitch that is machine readable as part of our solution to the problem. The program then finds all the possible solutions to the CSP we give it, and then converts that output back into the same format that has a one-to-one mapping with actual musical notation.

The outcome of this project was a resounding success. As we will describe in more detail later, we gave the program a pretty wide variety of different sample inputs to solve, and it was able to efficiently and accurately find solutions to all of them within the parameters that we set. We will also discuss some possible ways in which the current implementation could be even further expanded, but in the general case, our implementation meets all the goals we initially set out to achieve.

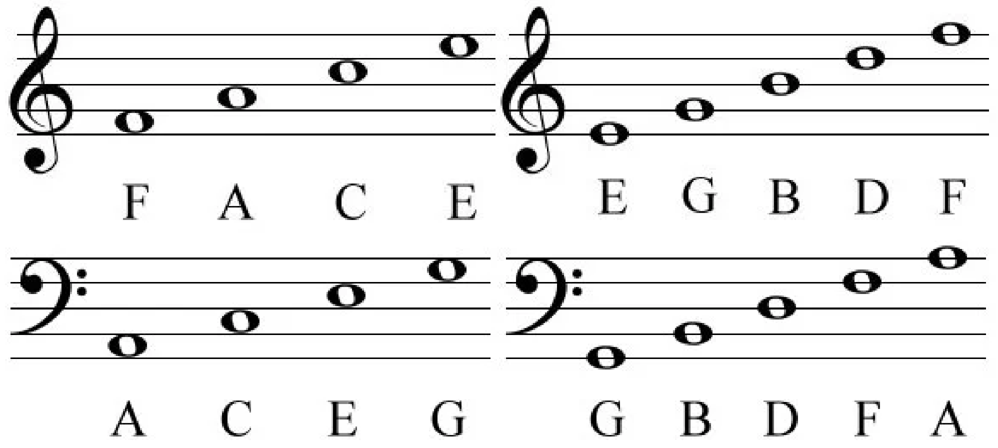
Background

The concept of four-part harmonies is a system that organizes chords, which is a group of notes played together, for four different voices ranging from the highest pitch range soprano, alto, tenor, and to the lowest pitch range bass. The pitches are how high or low the tone of a note is and there is a system that labels each note with its accompanying pitch called the scientific pitch notation. This notation system came from how the keys of a piano are organized, where each label consists of a letter that displays the note and a number that represents which octave the note is in. An octave is a set of eight pitches that has half its frequency with the next octave. So, for example, A4 and A5 are notes labeled using the scientific pitch notation, where both notes are of the same type but A5 has a higher pitch than A4 which is shown by the larger number. A mapping of sheet music to scientific pitch notation is shown below, from the public domain.



(https://en.wikipedia.org/wiki/Scientific_pitch_notation#/media/File:Scientific_pitch_notation_octaves_of_C.png)

Each note is labeled as a letter and there is an ordered list of notes that is repeated for each octave. The range of notes the human voice uses in the project in order is C, C#, D, E♭, E, F, F#, G, G#, A, B♭, B and the order repeats for each octave, which increases frequency of the pitches that produce a higher tone. With this ordered list of notes, it helps any viewer to read sheet-music by following the order of lines and spaces where the notes are placed. The graphic below showcases notes displayed in order showing which position each note takes on a music sheet.



(https://www.instructables.com/How-to-Read-Sheet-Music-for-Beginners/)

The top half of the sheet has the treble clef and represents the higher pitch range. The majority of its notes are sung by sopranos and altos. The notes placed on the spaces can be interpreted using the acronym FACE, which shows the order of the notes from left to right. For the notes placed on the lines of the treble section, there is a mnemonic readers use to remember the order: **E**very **G**ood **B**ird **D**oes **F**ly. The bottom half of the sheet has the bass clef and represents the lower pitch range sung mostly by the tenor and bass voices. For both the notes placed on the spaces and the lines, mnemonics are used to remember the note positionings. For the spaces the mnemonic is **A**ll **C**ows **E**at **G**rass and the mnemonic for the lines is **G**ood **B**oys

Do **F**ine **A**lways. Another symbol to consider are the ♭ flat and # sharp notes, which are located at the beginning of each music-sheet section. If a flat or sharp symbol is placed on a space or line, all notes on that space or line must be played in either flat or sharp until the end of that section. So if at the beginning of a section the first space of the treble clef there is a sharp symbol, all F notes are transitioned to F# for the entire section unless told otherwise.

Understanding how scientific pitch notation works and how to read notes on a music sheet are crucial skills to master in order to tackle the problem. It helps one recognize notes and pitches allowing them to perform more advanced music-theory practices such as four part voice writing.

Related Work

Using artificial intelligence to generate novel music, or to make music modeled after some given samples is an area of growing popularity and interest among musicians and computer scientists alike. As such, searches for literature on the intersection of music and artificial intelligence yield results almost exclusively of this nature. Using artificial intelligence for the analysis of Western classical music theory, however, is a much sparser topic. We were, however, able to find some literature relating at least tangentially to our goal.

In his article “An Artificial Intelligence Approach to Tonal Music Theory,” James Meehan discusses approaching musical analysis with artificial intelligence with a “linguist” approach, which he argues is quite similar to what (when abstracted) to tonal music theory, and thus natural language processing models could be used as a method of artificial intelligence doing analysis of music theory (1980, 61). The scope of what Meehan discusses here, however, is beyond what we aim to do, and involves analysis of classical music on a large scale, as a human might do, whereas we are trying to solve the comparatively simpler problem of specifically four part voice writing.

Somewhat similarly, a very interesting 2020 paper by Peter Harrison and Marcus Pearce looks into creating a computational cognitive model of voice leading which simulates how different voice leading choices affect musical perception from the listener (2020, 208-209). While this is very exciting research, and is somewhat closer to what we're interested in ("voice leading" being essentially analogous to "four part voice writing" in this context), it is still of a different, more complex scope.

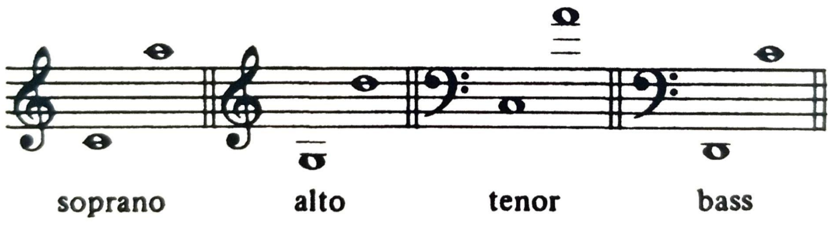
Let it be noted, however, that there is existing software that accomplishes nearly the same goal we're setting out to. For example, https://partwriter.com/ will take a harmony progression inputted by the user, and generate possible solutions according to the rules of four part voice writing. Similarly the software called Harmony Builder (http://www.harmonybuilder.com/) will provide "error" checking for four part voice writing done by the user. Neither of these are quite exactly what we're setting out to do, but they are very similar. And so, even if there is little to no formal literature/research on the topic, we do see examples of where this has been done, or at least attempted, in existing software.

Problem

Four-part writing has developed many conventions, but for this initial foray into the goal of solving four-part writing using a CSP, the number of conventions to follow was reduced. This strategy was employed to have a greater chance of having a functional project by the end of the semester. In particular, the four voices must be placed so that, for the entirety of the music:

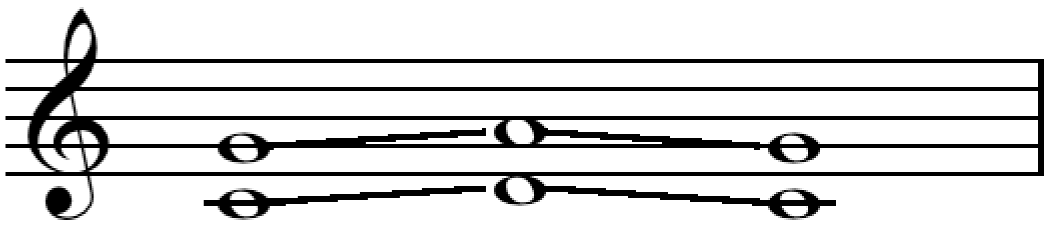
1. Each voice is in the accepted range.

   The accepted range for different voice types is based on the usual range for human voice types. In scientific pitch notation, a bass voice is able to go from C2 to C4; a tenor from C3 to A4; an alto from G3 to D5; and a soprano from C4 to A5. (Note that we adjusted the bass to have C2 as its lowest note, because it better aligns with scientific pitch notation).



2. No combinations of voices can have parallel fifths or octaves.

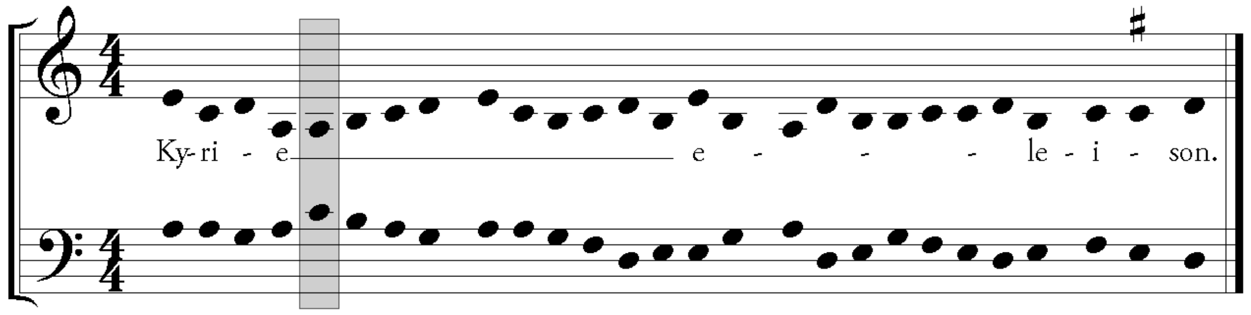
   The rules of four-part writing are defined to create "four distinct voices." Parallel fifths or octaves reduce the feeling of two voices being distinct, so they are not allowed. A parallel fifth is when the interval between two voices, subtracting any octaves (so a voice one octave and a fifth away is just being a fifth away), is a fifth on one beat and a fifth on the next beat while both voices change pitch. A parallel octave is the same concept but with an octave rather than a fifth. In the case of parallel octaves, the interval can be any number of octaves on consecutive beats. A sheet music example of parallel fifths is below, courtesy of Hyacinth on English Wikipedia.



(https://en.wikipedia.org/wiki/Consecutive_fifths#/media/File:Parallel_fifths_on_C.png)

3. Voices cannot cross.

   The ordering of the voices, from top pitch to bottom pitch, must always be soprano, alto, tenor, then bass. A voice crossing is then defined as when the ordering of the voices is not this order. For example, the alto pitch may be higher than the soprano pitch for a beat. A sheet music example of voice crossing, albeit with two voices rather than four, is below, courtesy of Michael Scott Cuthbert.



(https://en.wikipedia.org/wiki/Voice_crossing#/media/File:Kyrie_cuncti_ravenna_crossing.png)

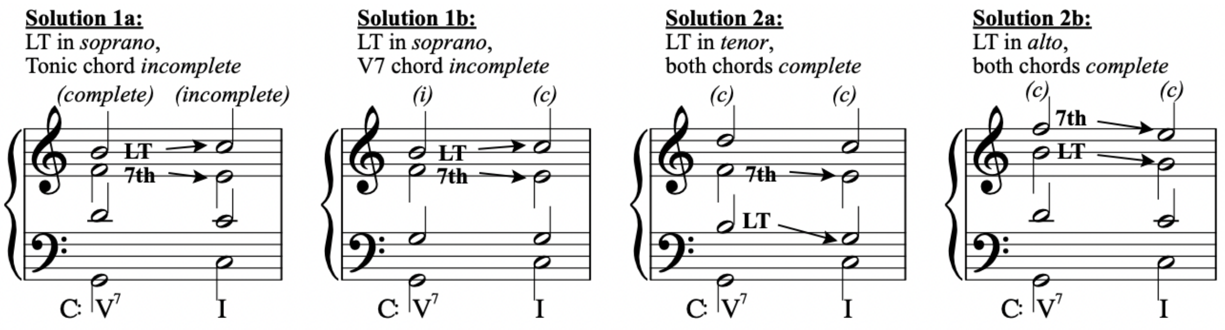
4. Each note of the harmony at each beat is present.

   Every beat has a harmony as defined by a chord and an inversion. A chord defines a set of pitches defined within it, and the voices at each beat will need to express each of these pitches. For example, C Major contains the pitches C, E, and G. There are too many chords that can exist to cover all in this description, but to know other translations, like that A Minor contains A, C, and E, we recommend outside sources. The inversion of a harmony defines what the bass note must be, as the bass note of a harmony contains a certain aesthetic feel attached to it.

   There is a caveat to this in that it is sometimes allowed to omit the fifth of a chord, but to align with our set of rules being simpler for an initial attempt at creating solutions, we will require the fifth at all times.

5. If the harmony has a chord with a 7th, the 7th must go down by step.

   Some chords are defined as containing a 7th. For example, C7 contains the pitches C, E, G, and Bb. Bb is the seventh in the chord because it is a seventh away from the root, C. Bb, therefore, must go down to either A or Ab depending on the chord that follows C7. A sheet music example of this is below, courtesy of Mark Feezell at LearnMusicTheory.net.

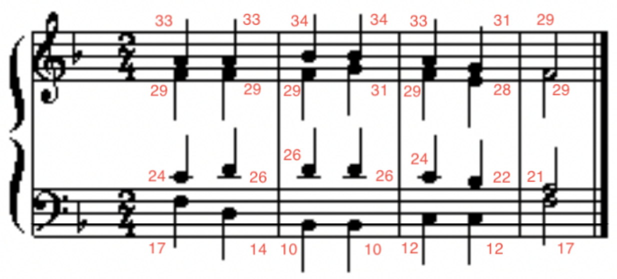


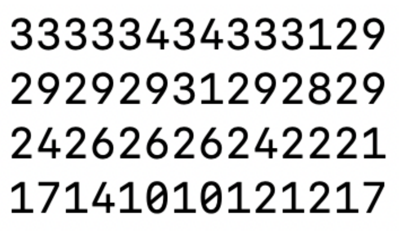
(https://learnmusictheory.net/PDFs/pdffiles/02-10-SATBPartWriting6SeventhChords.pdf)

Solution

As mentioned above, we decided to solve this problem by modeling it as a CSP. We took this approach because we felt that four part voice writing fits the parameters of a CSP really well. There are some known values and some unknown values, all of which have relationships with each other that place restrictions (constraints) on what values the others can be. In order to model part writing as a CSP, the first step was to make some machine readable format for the music.

To do this, we designed a four part voice writing file format with the extension .4pw that is a mapping from standard musical notation to a machine readable form. A 4pw writing file will always have 5 lines: one for each voice (SATB - soprano, alto, tenor, and bass), and the last one for the harmony. For each voice (line), the pitch of any given note is represented by a numerical digit in the form 00 (an integer with a leading zero if necessary), with 00 being our lowest note (C2 in scientific pitch notation), and each increment of one representing a half step. For example, 01 is C#2, 02 is D2, 12 is C3, 24 is C4, and so on. 00 is selected as our lowest note because it is the lowest note of the bass voice range. There is no need to handle notes lower than it. A piece of sheet music with each note marked with our numerical representation is below, courtesy of University of Tennessee and edited by us. How this then looks in a file is below, provided by us.





```
33333434333129
29292931292829
24262626242221
17141010121217
```

(https://music.utk.edu/theorycomp/courses/murphy/documents/PartWritingRules.pdf)

For the harmony line, harmonies are represented by three characters indicating the root, quality, and inversion of the chord. The root of the harmony is given by a number 0-11 (with A for 10 and B for 11) which again maps to the 12 distinct pitches. The quality is given by a single character that maps to one of the chord quality types, as shown in the table below. Lastly, the inversion is given by a character, given in the other table below. For example, 0Mr represents a C Major triad in root position.

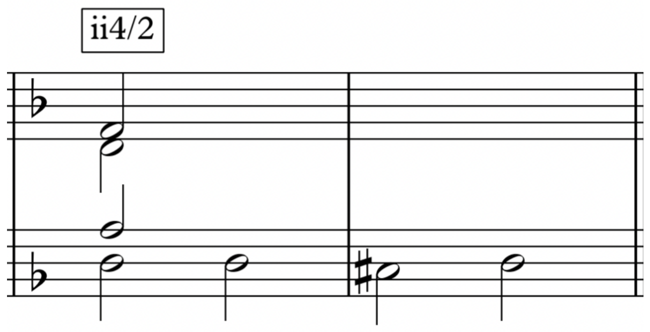
M	Major triad
m	Minor triad
b	Diminished triad
7	Dominant 7th
D	Fully diminished 7th
d	Half diminished 7th

r	Root position
1	1st inversion
2	2nd inversion
3	3rd inversion

Expanding the above example, a full file including the harmony line is below.

```
3333343433129
29292931292829
2426262624 2221
17141010121217
5Mr2mrAMr7m15M207r5Mr
```

However, the above example does not tell the whole story - it includes every note filled in. This is what a file would look like when it is printed out successfully by our program. An input to our program may look a bit different, with missing notes. We created a convention of using "??" to mean that the note in that line at that beat is missing. Below is an example of an exercise provided to us by Professor Rebecca Leydon of the Oberlin music theory department and its notation in our format. For fun, we also include a solution .4pw file from our program.



```
29??????
26??????
21??????
14141314
2mr4d39712mr
```

```
29434045
26343129
21282121
14141314
2mr4d39712mr
```

This file format is used as both the input and output of our program, and allows the program a way to parse musical notation. Our program is implemented in Python, and makes use of the library "python-constraint" which provides a CSP framework. The main program consists of "read_problem.py", "find_domain.py", and "rules.py". The first, "read_problem.py" is the main program, and handles input and output. It reads in a .4pw and parses it into a CSP by storing the known values (including harmony) in lists, and then adding all the unknown values as “variables” into the CSP framework provided by the library. These variables are labeled in the form “x#” where ‘x’ is a character from ‘satb’ corresponding to the voice, and ‘#’ is some integer corresponding to the beat (chronological placement) of the note in the music. For example, the first note of the alto voice is given by “a2”. The starting domains for each of the unknown values are then generated by “find_domain.py”, which involves parsing the harmony line and determining which pitches are possible. Then, all the constraints are added into the problem.

This is where the brunt of the work is done, as all of the musical conventions that exist have to be modeled as mathematical constraints/relationships between notes. There are relationships between notes and following notes, notes in different voices, and pairs of notes in different voices that must all be modeled. In “rules.py” all these relationships are modeled mathematically and added as constraints to the CSP framework. These constraints are defined as follows:

1. Voice ranges: simply, the pitches must be greater than or equal to the lowest pitch of that voice's range, and less than or equal to the highest pitch of that voice’s range.
2. Parallel fifths and octaves: taking two adjacent pairs of notes pair A and pair B, the difference between pitches in pair A and pair B must not both be 7 (parallel fifths) or 12 (parallel octaves), unless pair A and pair B are identical (the notes didn’t change).
3. Voice crossing: a given pitch must be less than or equal to the pitch of the voice above it, and greater than or equal to the pitch of the voice below it.
4. Harmony: at a given beat, the set of pitches made up of all four voices must be equivalent to the set of pitches required by the harmony.

5. 7th resolution: if a given pitch is the 7th of the harmony at that beat, then the following pitch must be either 2 or 1 less than the preceding pitch (resolving by step).

The “python-constraint” module also provides a solver for the CSP, which utilizes the min-conflicts algorithm as well as backtracking. This returns all solutions, if there are any, as a list of dictionaries mapping variables to their values. The program then parses these solutions and formats them back into the 4pw file format. In addition, there is a script “validator.py” which, given all the same rules/conventions, will return whether a 4pw is valid, and if not describe which conventions are broken. The main program can pass the solutions into this validator to independently confirm that the solutions follow all the musical conventions they’re supposed to.

Additionally, there is one other utility script “convert_pitch.py” which can be used to convert 4pw file formats into a human readable format by translating the numeric representations of pitch into scientific pitch notation.

Outcomes and Evaluation

To evaluate our implementation, we were generously provided a good number of samples of four part voice writing exercises from Professor Rebecca Leydon of the Oberlin music theory department. These example exercises covered a span of complexity from things given to Theory I students to Theory III level exercises.

The program worked phenomenally! It was able to solve all of the samples that we threw at it. The way we evaluated whether the solutions it gave were correct – and by extension the way we evaluated the success of our implementation – was with the validator program described above. The validator program worked completely independently of the CSP framework that the main program used, and so we could have an accurate judge of the validity of the solutions.

Our program worked in everything that we set out for it to achieve at the beginning. It solves the samples correctly within the bounds of complexity that we laid down at the beginning. One thing that could be refined is the output. The program finds all possible solutions, but we could implement a way to rank/order these solutions, which we will describe in more detail later. There could also be expansion of complexity, implementing more constraints to allow the program to accurately solve more advanced musical problems. Some examples of this will also be given later. But overall, the program works great. It is efficient in finding solutions (the example used in this text takes 0.135 seconds for the solution to be generated), the solutions are valid, and it can solve all the problems we defined as our origins goals.

Conclusions and Future Work

Motivated by personal experiences as Conservatory students studying music theory, we set out to create an intelligent program that could complete four part voice writing exercises automatically. To do so, we created a machine readable file format to represent standard musical notation, and then modeled the writing exercise as a Constraint Satisfaction Problem. Solving the CSP with our program provides musical solutions to the exercises given as input that follow all the given conventions of four part voice writing. Our implementation was a resounding success, and was able to successfully solve all the sample problems we prepared, provided by an Oberlin music theory professor covering a span of different complexities from Theory I to Theory III.

Although our implementation was able to successfully solve all these examples, there is certainly more work that could be done on the program to expand it even further in the future.

For example, there are more complicated harmonies (augmented 6th chords, major 7ths, suspensions, and etc.) that aren't implemented in our program and were not in the samples we had it solve, but could very well show up in even more advanced part writing exercises. Furthermore, depending on the exercise there can be dozens if not hundreds of possible solutions. And even though they're all technically valid by the conventions we implemented, some are better than others in a way that can't be modeled by a CSP. A heuristic could be developed to sort the solutions the program gives by how "good" they are from a musical perspective. Some possibilities included a "minimum distance moved" heuristic, judging by how little each of the voices moves in the solution. But there are many possibilities here, all heavily reliant on theory knowledge to create some objective standard of a "good" solution.

Even further, this program holds the untapped potential to actually generate completely novel music subject to the conventions of four part writing and given a harmony line. It's perfectly valid in the current implementation to give it an input where all 4 lines are unknown, in which case it would use the harmony given to generate all the possible (which would be a *lot*) musical samples in the standard four part writing style. Certainly there are lots of exciting prospects in ways this program could be expanded beyond its current use!