# Genetic Programming Framework For Schenkerian Musical Analysis

Thomas Waters
University Of Vermont
Burlington, VT
twaters@uvm.edu

Dave Landay
University Of Vermont
Burlington, VT
dlanday@uvm.edu

Arlo Cohen
University Of Vermont
Burlington, VT
acohen32@uvm.edu

## ABSTRACT

Schenkerian analysis is a method of structural tonal musical analysis which seeks to find dependencies within a musical composition. This type of analysis reveals the hierarchical structure behind a particular score which reveals the particular function that each note plays in relation to the composition as a whole. A way of representing this analysis is to construct a tree that represents all of the fundamental structure as well as all the elaborations and prolongations from the fundamental line that are needed to reach the final score. We present a new way of constructing and searching for potential solutions. Our framework uses genetic programming to evolve a potential set of fundamental ideas and elaborations that can be used to explain the compositional structure of music score. To be able to guide the algorithm towards the actual score we are analyzing, we used a version of an edit distance similar to a string distance metric called the levenshtein distance. This fitness evaluation allowed us to iteratively grow a tree towards a correct solution of the input piece.

## CCS CONCEPTS

• **Applied computing** → **Sound and music computing**; • **Computing methodologies** → *Genetic programming*; • **Theory of computation** → *Evolutionary algorithms*;

## KEYWORDS

ACM proceedings, Musical Analysis, Genetic Programming

**ACM Reference Format:**
Thomas Waters, Dave Landay, and Arlo Cohen. 2019. Genetic Programming Framework For Schenkerian Musical Analysis. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Over the years there have been many different techniques for performing musical analysis. Most of these techniques focus on tonal music. Tonal music is the most popular music style today and it is focused around the the western idea of the diatonic major scale. Structural musical analysis is the technique of finding relationships of different components within the musical score as a whole. These relationships can be how neighboring notes relate to each other in terms of embellishments of chords which played in relation to background ideas of a piece. Schenkerian analysis is one of the most well developed types of Schenkerian analysis. Schenkerian analysis is a hierarchical analysis which is concerned with finding the "interrelationships among melody, counterpoint, and harmony". [1] It was developed to understand the music created from the common practice period (1600-1910) from composers such as JS Bach, Haydn, Mozart, Beetoven, Schubert, Chopin, and Brahms. It is used to show how a score moves from compositional simplicity to complexity.

Schenkerian analysis and more specifically the works of Lerdahl and Jackendoff [7] lend themselves well to analyzing music in a computationally significant manner. Musically significant notes are notes which are articulated in performance. Compositional or structurally significant notes are notes which are used to derive the purpose of other notes in a score. Schenkerian analysis tries to find the relationships and relative importance in terms of their structural relationship in the music. This technique is generally done as a reductionist approach where you start with the score and work towards higher levels of abstraction to try to understand the hierarchical relationship between all the notes in a piece. Working in this way, the notes that appear only at the lowest level of abstraction are the least significant structurally, where the notes that remain until the end of the analysis are the most important to the overall piece. This type of analysis is supposed to mirror and reconstruct one of the ways in which the composer might have constructed the score. This is done by starting with the Bassbrechung or the I-V-I bass line progression played over the Ursatz which is the 3-2-1 descending melodic line in the respective starting musical key. Starting with that structure, the rest of the score is represented as a series of embellishments from this structure to create more content within the respective score.

A student would of music would perform Schenkerian analysis to get a better understanding of music theory and a higher level of understanding of musical composition. Automating this technique could provide feedback to the student for possible correct solutions to a musical score. A performer might find it beneficial to use this type of analysis in order to better understand the score they are trying to perform. This helps with memory techniques such as chunking to remember hierarchical layers of information for faster and easier recall during performance. Theorists may also find this information interesting when trying to study the changes in computational techniques used by individual composers, or for understand structural changes in musical composition over time. This type of analysis can also be used with musical software applications for


*Conference'17, July 2017, Washington, DC, USA*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

classifying songs within groups of that are structurally similar. A possible application could be Musical Information Retrieval (MIR).

This paper proposes a new method for analyzing a score using the techniques of Schenkerian analysis. This approach uses an evolutionary learning technique called genetic programming where an analytical solution is grown generatively by evolving a set of rules which could produce the score that is being analyzed. When these individuals are able to properly recreate the input piece, the result is a hierarchical tree of rules that can be used to generate the given score. The tree representation represents the Schenkerian analysis. This technique allows the analysis to take advantage both the fields of generative music as well evolutionary computation to solve a problem which has commonly be solved as a reductionist search problem.

This paper starts by covering the other state of the art techniques which have been developed to computationally analyse scores of music using Schenkerian techniques. It then goes over the methods used to do this analysis, which includes the pre-processing, tree representation, musical operators, as well as the selection, crossover, and mutation operators needed to perform the genetic programming. The paper then goes over how well this technique was able to perform in this search space using the different genetic programming techniques. Lastly it goes over some different directions that this framework can be extended to get better performance as well as how to extend it to be able to analyze a wider range of musical scores.

## 2 RELATED WORK

Music is experienced relative to the interpretation of the listener, and in parallel, musical analysis is up to the interpretation of the analyst. Each piece of music has multiple correct analyses, so algorithmic interpretations will never be exhaustive [5].

Furthermore, as Marsden warns, Schenker never truly formalized his analysis himself [9] . Many "correct" Schenkerian analyses could be argued for one piece of music, and all could be descriptive of different listener's experiences, or different musician's delivery. Despite these obstacles, many works have been done on generative algorithmic analysis. Several provide means for multiple interpretations, an important step in the right direction.

Marsden's works use a reductionist approach. An Automatic Derivation of Musical Structure finds a path on a dynamic programming matrix of the individual reductions from the original score to the schenkerian analysis. [9] Another paper moves toward our approach, using operational reductions instead of individual. [8] A paper from the University of Alicante details another approach, using weighted directed graphs to represent harmonic key changes, outputting a roman numeral analysis. They analyze a corpus of music to diminish the search space based on the genre, for example, not allowing jazz chord progressions while analyzing a baroque piece. This work was also significant as it used a genetic algorithm to train the weights in the graph. Although they use evolutionary computation for schenkerian analysis, the means of building the final tree are very different, as they use a directed acyclic graph.

With such a large search space, a semi-automated guided approach offers an analysis catered to the user's vision, at the cost of user interaction. Marsden offers such software that is an extension of his previous works. [10]

Using a directed acyclic graph, a question of importance is how to search. Marsden and Wiggins use an AI search among other search strategies to better navigate the search space. [11]

Lerdahl and Jackendoff are widely referenced and provides a framework for decomposing a piece into its theoretical parts [7]. The paper describes sections using trees, to connect the piece together to one root, and to maintain the relative nature intrinsic to varying key signatures. The use of trees was inspirational to the use of genetic programming in our algorithm. Using (evolutionary computation/genetic programming) to create a variety of trees, we aim to capture different Schenkerian musical analyses.

## 3 METHODS

### 3.1 Test Score

As an initial first test of the system, it is useful to try evolving a musical score over simple pieces. For this task, we chose *Hot Cross Buns* as it is short in length, and low in melodic complexity.

### 3.2 Score Pre-processing

To generalize our methods to work within the Schenkerian framework, we perform some initial pre-processing steps to prepare a ground truth musical score for analysis. First, we ensure that all notes in the score are of the same duration. Next, we enforce that there are no melodically duplicated notes or rests. This is because Schenkerian analysis is only concerned with melodic progressions and not particular implementations of rhythm. In addition, rests are removed and note duration is not considered in the comparison of a score and a resultant analysis. We assume that by removing abstractions and complications from a ground truth score, that it will be easier to evolve an accurate analysis with genetic programming by reducing the search space, and make it unnecessary to introduce a more complex rule set. Furthermore, our framework constructs a composition from the chromatic scale. The chromatic scale, or twelve-tone scale, consists of twelve semitone pitches which are equally spaced half-step intervals from each other. Since each tone is equidistant from it's neighbors, the scale has no underlying tonic or key. Hence, we make no assumptions about the root key and do not have to manually assign one.

### 3.3 Genetic Programming

The genetic program uses a hierarchical tree representation to perform the analysis. The root node of the tree is a key signature operator which turns a series of relative operations into a score based off the fundamental note of that key signature and tonal center. For example, a song in the key of C major would be based off of the C4 key, which is at the center C key on a standard piano keyboard. An operator works by translating notes in a geometric integer space, where each note is represented by an integer value in pitch space. Pitch space is linear, and is used to transform tones from the non-linear frequency space that the human ear perceives. This helps to simplify any analysis which is done within this pitch space.

The solution to an analysis is the resultant tree structure, assuming that the tree is able to produce the input base line score.

The hierarchy of operators present in the tree embeds information about the composition structure and the embellishments present in the piece of music that is being analyzed. Genetic programming is often used to train trees that are good at predicting some outcome given some input. It can also be used to determine which variables are relevant to the problem being solved, as a kind of variable reduction method. Our approach to growing trees differs from traditional genetic programming; there are no input variables, and we are trying to solve a known solution. The goal is to construct a tree that can accurately represent a score that is given. additionally, unlike most genetic programming methods, there is no global set of meta-parameters that conform to all analyses. Hence, an analyst must tune parameters specific to each score that will be analyzed. These include features like the length of the piece, which will increase the search space.

## 3.4 Tree Representation

The representation of an individual in this framework is a tree that contains a root key signature and a set of operators. The leaf nodes are also considered operators in this representation, and there is only a single kind of leaf node. That is the identity translation (a translation of zero). This could be extended to support greater functionality but provides a framework for exclusively using referential operators instead of working in a key signature specified by the analyst. All notes are considered in terms of their structural relationship as a tree is evaluated from leaves to the root node. This allows low level musical embellishments to be independent of both the high level structure, as well as the particular key signature that the composition is written in.

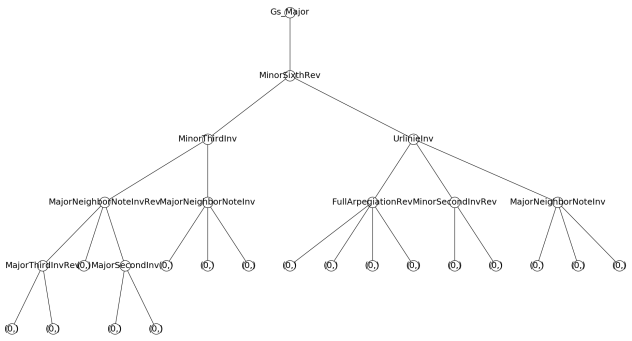


**Figure 1: An example of a tree representation of a musical score. Specifically, this tree is a proposed analysis of Bach's *Gott sei uns gnädig und barmherzig*, BWV 323 and shows the construction of a composition from a root key signature.**

In **Fig.1**, the nodes represent various translational operators that are typically used to describe the decomposition of a piece of music in a Schenkerian analysis. Each layer in the tree demonstrates how musical operators are applied to the root key signature to reconstruct a ground truth musical score. If a score is perfectly represented by the order and carnality of the set of leaf nodes, then we say that the tree is an accurate representation of a score's decomposition in the Schenkerian framework.

## 3.5 Musical Operators

Node operators are a set of translational operators that can be represented as a series of offsets in the chromatic, 12 note space. For example starting with the note C, applying the transnational operator of a minor neighbor note, which has a translation representation of (0, 1, 0), will result in the series of notes (C, C♯, C), where C♯ is a single semitone (one step) above C. Many of these operators are generic operators taken from Schenker's approach to tonal analysis, and are compatible with the more simplistic hierarchical tree representation of music. The list of all these base operators can be seen in table . These operators can also come in three different forms depending on whether the new operator is unique or not. These include the reverse of an operator, which reverses order of the original operator, and an inverse operator, which is the array where each element is subtracted by the repeating musical interval of 12 semitones. As an example of the latter, the inverse of the arpegiation (0, 4, 7, 12) is (-12, -8, -5, 0). The final additional operator is the inverted reverse of the original. This creates up to 4 possible operators from the initial base operator, assuming that the variants are all unique.

Every operator has a single parent so that it fits into the simple hierarchical tree structure. An operator has a number of children elements equal to the number of translational elements it contains. The example arpegiation above would have have four children elements. These operators are evaluated by taking the children elements which would be a list of relative offsets, and then applying the offset from that operator to each of the children from the respective location in the operator. Using the above arpegiation as an example. This operator could take a couple of children embellishment elements like (0, 1, 0), (0), (0), (0, 1 0). The operator has an offset vector of (0, 4, 7, 12), then each of the children are translated one at a time corresponding to that index in the operator. After doing this evaluation the resulting vector would be (0, 1, 0, 4, 7, 12, 13, 12). This process continues until this score is evaluated by the root operator actualizes a series of relative offsets in a particular key signature. For example if this vector was evaluated in the key of C major, the resulting score would be (C, C♯, C, E, G, C, C♯, C) where the last series of (C, C♯, C) is an octave (12 notes) higher than the original C in the piece.

## 3.6 Fitness Function

The leaf nodes of a tree are the notes which form the proposed musical score. We assume that any tree representation with a set of leaf nodes (notes) whose order, assignment, and cardinality match the ground truth musical score exactly, is an accurate model of musical composition in the Schenkerian framework. Hence, we may treat a proposed score and the ground truth as two strings of musical notes, and compare them using any string-distance metric to assess the accuracy and quality of an individual tree. We use the Levenshtein distance metric, a dynamic programming string alignment algorithm, to assess the fitness of a set of leaf nodes, and seek to minimize the distance between a score and a ground truth piece of music. Our version of the Levenshtein distance fitness metric punishes for gaps with a variable penalty and penalizes incorrect pitches by their pitch distance. Utilizing a distance metric as measure of fitness provides several benefits. Mainly, we can encode

| Operator | Translation |
|---|---|
| Bassbrechung | (0, 7, 0) |
| Predominant II | (0, 2, 7, 0) |
| Predominant IV | (0, 5, 7, 0) |
| Arpegiation | (0, 4, 7) |
| Full Arpegiation | (0, 4, 7, 12) |
| Register Transfer | (12) |
| Urlinie | (4, 3, 0) |
| Minor Neighbor Note | (0, 1, 0) |
| Major Neighbor Note | (0, 2, 0) |
| Passing Note | (0, 1, 2) |
| **Intervals** | **Translation** |
| Minor Second | (0, 1) |
| Major Second | (0, 2) |
| Minor Third | (0, 3) |
| Major Third | (0, 4) |
| Perfect Fourth | (0, 5) |
| Augmented Fourth | (0, 6) |
| Perfect Fifth | (0, 7) |
| Minor Sixth | (0, 8) |
| Major Sixth | (0, 9) |
| Minor Seventh | (0, 10) |
| Major Seventh | (0, 11) |

**Figure 2: The base operators used for relative translations of notes to create the final score. There are up to 4 possible solutions if all the reversed and inverse variations produce new unique operators.**

the length of the ground truth piece of music we are analyzing and limit the pitch distance between notes in the fitness function itself. This avoids solving a multi-objective problem directly and penalizes individual trees that grow too large.

### 3.7 Levenshtein Distance

This type of edit distance emphasizes preserving correct information. It is implemented iteratively, filling an array of all possible alignments between two strings, with the optimal alignment score at the bottom right. Each step takes the minimum distance from offering a deletion, offering an insertion, or taking both characters as is and penalizing for a mismatch if so.

### 3.8 Parent Selection Mechanism

After individual fitnesses are calculated, the parent selection mechanism chooses members from the current population to mate; creating new offspring in the next generation. We use tournament selection as our mechanism in this study. In tournament selection, the analyst specifies a competition size $k$, which indicates the number of individuals that are selected uniformly at random to compete with one another. At each generation, the most elite individual will be selected to help seed the new population. Tournament selection is a widely used mechanism because it allows an analyst to have control over the amount of selection pressure enforced on the population. This is controlled for by selecting a reasonable tournament size $k$. This feature of tournament selection is important in our application because as the sizes of individual trees in the population begin to converge, less diversity can be introduced into the population, and the evolutionary process will converge on sub-optimal solutions.

### 3.9 Crossover

Single point crossover is applied to control selection pressure. With probability $P_c$, a single terminal node is selected between two parent individuals. Two offspring are formed by swapping the subtrees, whose root nodes are the selected chromosome, from the two parent programs. Individuals that

### 3.10 Mutation

Each musical operator is treated as an equal contributor to the fitness of a composition. In other words, we consider the musical operators as a set of uncorrelated transformations so as not to explicitly define the rules of composition, nor bias the evolutionary process towards a particular set of operators. Hence, we employ uniform mutation over the terminal nodes of an individual, whereby a new operator is assigned to a node uniformly at random with probability $P_m$.

### 3.11 Survivor Selection

The survivor selection mechanism is performed after mutation, and determines which of the offspring trees will form the population in a subsequent generation. Eiben et. al [4] explain two paradigms of survivor selection, or evolutionary strategies, called $(\mu + \lambda)$ and $(\mu, \lambda)$. $\lambda$ denotes the offspring created via the process of mutation; the best $\mu$ of which are chosen deterministically from only the offspring, $(\mu, \lambda)$, or from both the parents and offspring, $(\mu + \lambda)$ to re-grow the population. A recommendation by the authors is to scale $\mu$ to $\frac{1}{7}$ of $\lambda$ due to high selective pressure. We utilize the $(\mu, \lambda)$ paradigm as our replacement mechanism as all parents are discarded and has the potential to shrink the size of local optima.

## 4 RESULTS

### 4.1 Tuning Parameters

To obtain accurate analyses, it is important that certain meta-parameters of the system are tuned to allow for adequate selection pressure, while maintaining a high diversity among the population. In particular, good configurations of features like the crossover and mutation rates, as well as the selection and replacement criteria are crucial for solving tasks in complex search spaces, as is the case with music decomposition.

To find good parameters, a naive grid-search over sets of crossover and mutation rates, as well as a set of upper bounds for the maximum allowable tree depth and number of generations was performed. The search for optimal crossover and mutation rates were over the sets $[0.1, 0.2, 0.3, 0.4]$ and $[0.1, 0.05, 0.001]$ respectively. As $(\mu, \lambda)$ survivor selection was implemented, we found that a crossover rate of 0.4 in combination with a high mutation rate of 0.1 yielded the best initial results.

Given that the length of the ground truth score was known, the sweep over the maximum tree depth was limited to the range $[5, 7]$. This fact helped us to avoid unnecessary computation, but allowed us to explore a large enough part of the search space to ensure some

genetic diversity. Trees grown to a maximum depth of 6 proved to satisfy this notion.

For each combination of the above, we tested over the set of evolutionary run times [50, 100, 300] to determine how many generations the average analysis would take to converge to a low-fitness local optima. A run of 100 generation proved to be an adequate amount time for this to take place.

These parameters were initially tested with simple genetic algorithm with parent selection over the entire population. A set of tournament sizes, [5, 10, 15], was also explored to determine the degree of selection pressure in the system we determined that 5 was the upper limit on tournament size. Finally, survival selection was added after the sweep to improve upon the results yielded from the set of good parameters. For an initial population size of 2000 individuals, we found that $\mu = 215$ and $\lambda = 1750$ to be the most performant parameters for survival selection, under the $\mu + \lambda$ paradigm.

In summary, the parameter sweep indicated that the following parameters were best suited for our task.

| Parameter | Value |
|---|---|
| Max. Tree Depth | 6.0 |
| Crossover Rate | 0.4 |
| Mutation Rate | 0.1 |
| Tourn. Size | 5.0 |

**Figure 3: Results of the parameter search**

## 4.2 Gap Penalty

With these tuned parameters, we ran an experiment to test the effect of a shrinking gap penalty on fitness. We hypothesized that a decrease in the penalty would lower the selection pressure, and therefore lower fitness. In section **4.2**, we show the results of this experiment, given a penalty of one up to six. The results of **Figure 4** serve to demonstrate two things; that the maximum size a tree can grow is bounded, and that the average minimum fitness is lower for smaller gap penalties. **Figure 5** demonstrates that we could not reconstruct the ground truth score perfectly within the rules of the framework.

## 4.3 Evaluation of a Simple Musical Score: Hot Cross Buns

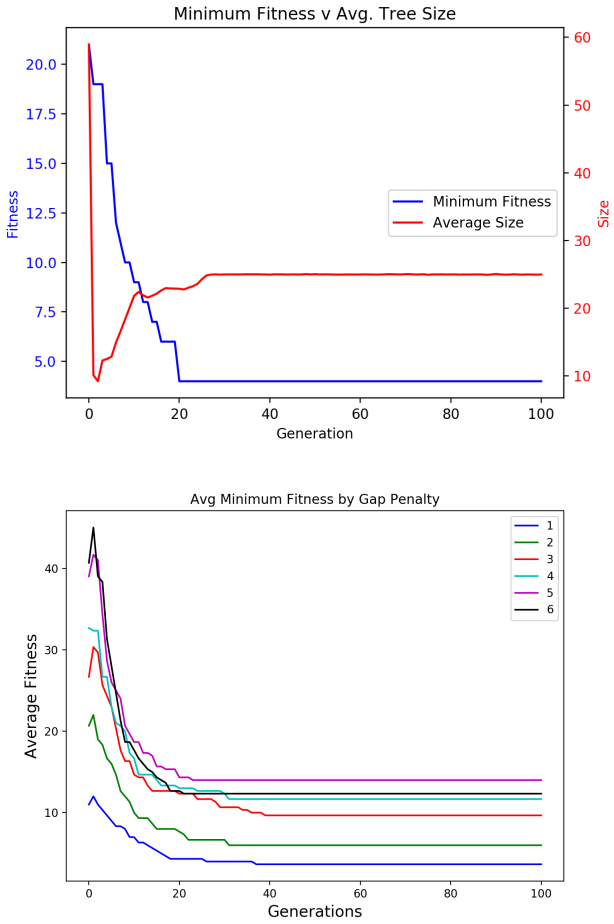


**Figure 4: (Top) The fitness curve and tree size for the best found individual with a gap penalty of 1 over $g = 100$ generations, and out of $N = 3$ samples. (Below) The average minimum fitness over $N = 3$ sample run for $g = 100$ generations each. The numbers in the legend denote the gap penalty assigned at the onset evolution.**

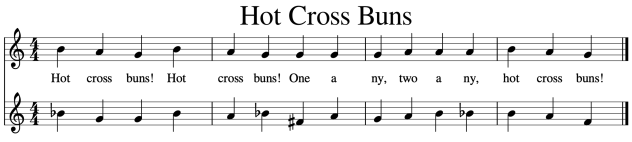


**Figure 5: The composition of *Hot Cross Buns* from the best individual in the run. The top staff is the ground truth composition and below it is the evolved score.**

# 5 DISCUSSION

Our results show the importance of the gap penalty, as well as the relationship between fitness and tree size. However, several trial runs conducted independently of the experimental results in **Figure 4** revealed that too low of a gap penalty will not guarantee that a tree is grown to full size; i.e: that the identity operators in the set of leaf nodes representing the predicted notes will be fewer in number than the number of notes in the ground truth piece. Hence, we were lucky that our final representation grew to maximum depth.

We weren't able to converge on a correct solution which would yield the input piece. As we see in figure **Figure 4** we have problems with premature convergence. This is probably due to having too high of a selection pressure which is reducing the possible search space too early. We need to be able to find more ways to reduce the selection pressure to determine if this is the true cause of the programs inability to converge on a correct solution.

An important benefit of this framework is that it is highly extensible. This will be expanded upon further in the future work section. This is imperative especially when studying a domain such as music. Music by it's nature is highly subjective so people are able to produce many works that don't follow Schenker's proposed true nature of music. This apparent when you look at the works of atonal (music with no clear central note) composers such as the modernist Prokofiev and the surrealist Schoenberg. Although Schenker was concerned with representing tonal music, there is a spectrum between tonality and atonality. This is also only one example in a way in which music can have a large variety in computational strategies. Having a framework that is extensible is important to be able to tailor the algorithm to the problem domain.

One issue for complete automation of this solution is to try to understand the validity of a Schenkerian analysis. This type of analysis is notorious for having many traps for human analysts to stumble into. Even if they are able to make it to the end with a potential solution as we are trying to do, not all solutions are created equal. Our framework attempts to give us some possible analyses which it purports to be correct solutions. Unfortunately, not all solutions are as valid as others. A person trained in Schenkerian analysis would be able to determine which solutions are better than another. There doesn't exists many methods of determining the quality of an analysis, and there doesn't exist any methods of determining the quality for a tree representation. Depending on the problem being solved, this may be something that may need to be solved before this could be used in implementing a solution.

# 6 FUTURE RESEARCH

## 6.1 Harmonic Analysis

This framework is only a basic structure for solving for a correct solution to Schenkerian analysis. Firstly, this technique currently analyzes only the melodic progression through time but doesn't cover any of the vertical dimension or the harmonic progression. This would require a new set of operators and relationships to be created to support the multitude of new possible interactions that come with adding in the other dimension. This is also going to change how the fitness needs to be evaluated, since the distance of the individual would then depend on that of the second domain as well. This is a non-trivial problem, like determining the pitch distance, because of the way people hear notes that are played at the same time as well as how chord and harmonic progressions are perceived over time. This does not have a direct relationship to the distance in the frequency space like pitch distance does. Haas, Wiering, and Veltkamp propose a method of determining harmonic distance by accounting for the musical relationships between two chords by comparing their distance in a harmonic tonal pitch space [2]. Using this method, the levenshtein distance could be extended to handle this new type of data. An important parameter to tweak here would be the deletion penalty because it could have a large impact in the fitness search landscape and on how the algorithm performs.

Another thing to consider when extending this framework into the second dimension is that when processing, it may be beneficial to process the bass and melodic musical ideas separately. These two musical ideas may behave differently and may have a different set of operators that may apply to each. This could help to reduce the search space by cutting this problem into two distinct sections. However, these sections are not independent. There would need to be a way to keep track of the interactions between the bass line and the melody.

## 6.2 Directed Acyclic Graph Representation

Up to this point, the analysis process has made the assumption that the music under analysis is purely hierarchical. This is supposing that all the structure can be derived from the notes directly above itself in the hierarchical tree. This isn't the case, especially when you are looking at music which is polyphonic (a song that has multiple voices), or the difference between a melody and the same melody played in a round. To be able to account for this This method could be extended to use a Directed Acyclic Graph (DAG) format. This is a tree structure that can have one direct parent as before, but can have multiple other secondary, tertiary, etc. parents that have other relationships to the operator in question. Adding in this tree structure would allow this algorithm to be able to solve a larger repertoire at the cost of a larger search space. This would also require a different tree evaluation method, as well as new mutation and crossover methodologies to be implemented.

## 6.3 Operators

The musical operators used in this paper are by no means the full set of operators devised to perform Schenkerian analysis. So for starters, implementing a larger set of operators would be important if a piece in question was using those technique for creating their composition. Secondly, the operators that are implemented are only the ones that are single parent operators. Many of the operators used in this type of analysis are used to fill in information in relation to multiple notes which are not necessarily neighboring notes.

One of the ways to achieve this more rich set of analyses was to implement the DAGs. This also means that the operators that are created would have to be created with this in mind. Operators would have to be created to handle multiple parent nodes. This is especially apparent when dealing with musical scores that have polyphonic melody lines where these different melodies have interplay between each other.

When this technique is extended into the harmonic domain to encompass multiple notes being played at the same time the operators are also going to have to be modified to handle this behavior. This is necessary to be able to handle both polyphonic melodies as well as the interactions of chords for harmonic progressions. When handling chords, the operators are going to have to be built to have a greater understanding of the interactions of harmonies between individual notes as well as the harmonic progressions that are built into a piece of music.

As these layers of complexities are added onto the operators, the ideas of high-level concepts and low-level concepts are going to be something that will most likely be important to either segregate or reward for having particular operators at different depths in the trees. The biggest thing to pay attention as the complexity of the operators increases would be the dramatic increase in the potential search space created by this operator space.

### 6.4 Genetic Programming Improvements

One of the things about music is that it often has a lot of repetition. Repetition is something that is comforting to people and helps to drive home particular musical ideas as well as make strange ideas sound familiar. Since this is the case this algorithm techniques of meta operators creation [3] which would help to evolve sub-problems which could be reused to create different types of embellishments within the analysis which would help especially as the length of the score increases.

By nature of this problem, multiple good solutions are possible and encouraged. The more correct and unique solutions that this algorithm is able to produce, the better the beneficial the program would be to a user. This means that not only the accuracy of the search is important, but also the variety of the resultant solutions are important. Because of this criteria, something that can maintain a more unique set of individuals would be beneficial not just for genetic diversity but also for the set of possible output solutions. Techniques like novelty search [6] could be incorporated either directly or through the use of some kind of multi-objective search criteria could help to achieve that end.

There are other possible fitness functions that might have better fitness landscapes for this type of evolution as well. Using the levenshtein distance is only one way of measuring the distnace between to different scores. It is hard to tell how deceptive this landscape may be. This fitness function would also take a lot of work to improve into something that can handle two dimensions of data through the addition of harmony notes as well. The search method employed by the genetic algorithm also does not necessarily follow a linear search method through the landscape. Depending on which subtrees are swapped and if operators high in the tree, it can have a large impact on how the individual travels within the firness landscape.

### 6.5 Generative Music Generation

What is also exciting about this framework is that it is extensible to another sub-domain of music. The technique generates trees which are evaluated to create a score of music which has a fitness criteria to generate a piece of music which is supposed to match the input score that we give it. This fitness criteria is imposed because of the problem we are trying to solve. This framework is just as useful for creating new music that is evaluated by some type of more arbitrary fitness function. This means that this framework can be used to generate things that are unique which match some type of criteria that are set forth by the user. Through the control of both the fitness function and the available operators, the artist would be able to create new music that would then start to conform to some sort of idea that they can specify in more abstract terms and let the algorithm find something that meets their criteria. Since this wasn't the focus of the paper, this idea wasn't pursued in the scope of this project. However, this does not preclude this framework from also being used in this context. Many of the other topics mentioned in this section also pertain when extending to this framework also apply to using it as a tool for creating generative works of music.

## 7 CONCLUSION

We present a new framework for creating a schenkerian analysis which uses genetic programming to search for potential solutions. This framework takes advantage of similar tree and operator frameworks that have been laid out in other works, but provides a novel way for searching for the correct solution. The solution matching criteria using levenshtein distance was also proposed as a possible fitness landscape to guide the genetic program towards the correct solution. The levenshtein distance is able to allow for gaps between notes, as well as accounting for the distance between incorrect pitches.

The current state of the framework is not able to completely converge to a possible correct solution. This leaves room for improvement before moving on to extending this framework to be able to encompass a larger variety of input works. This could be largely due to premature convergence, which can be seen when we vary the gap penalty. New approaches to account for selection pressure can help to relieve this issue.

One of the benefits of this method is that the framework is highly extensible. There are multiple directions of potential areas where this technique could be researched. This technique can be used to solve a much larger set of music by allowing for the tree to be represented with multiple parent connections with DAGs and allowing the framework to extend into the vertical harmonic domain. Also, since this framework is built off of the evolutionary technique of genetic programming. This allows us to take advantage of all the work that has been done in this field to provide inspiration for possible directions of future work. Examples of this kind of extension includes using meta-operators to take advantage of the level of repetition which is common within music or uses of multi-objective search can help guide the search in beneficial directions. This technique wasn't able to provide the initial results to solving the given input problems but provides a framework that could be used if the potential stumbling blocks are removed. This framework also provides a lot of room for growth if it proves to be successful approach for solving Schenkerian analyses.

## REFERENCES

[1] Allen Cadwallader. 2006. *Analysis of Tonal Music: Schenkerian Approach.* Oxford University Press, Oxford, United Kingdom.

[2] W. Bas de Haas, Frans Wiering, and Remco C. Veltkamp. 2013. A geometrical distance measure for determining the similarity of musical harmony. *Int J Multimed Info Retr* 2, 3 (Sept. 2013), 189–202. https://doi.org/10.1007/s13735-013-0036-6

[3] Bruce Edmonds. 2001. Meta-Genetic Programming: Co-evolving the Operators of Variation. *Turkish Journal of Electrical Engineering & Computer Sciences* 9 (2001).
[4] Agoston E. Eiben and James E. Smith. 2010. *Introduction to evolutionary computing* (1. ed., corr. 2. printing, softcover version of original hardcover ed. 2003 ed.). Springer, Berlin. OCLC: 837781639.
[5] Christy Keele and Todd McCready. [n. d.]. Automated Schenkerian Analysis. ([n. d.]), 7.
[6] Kenneth Lehman, Joel. Stanley. 2008. Exploiting Open-Endedness to Solve Problems Through the Search for Novelty. *MIT Press* (2008), 329 – 336.
[7] R. Lerdhal, F. & Jackendoff. 1983. *A Generative Theory of Tonal Music.* MIT Press, Cmbridge, Massachusetts.
[8] Alan Marsden. [n. d.]. Towards Schenkerian Analysis by Computer: A Reductional Matrix. ([n. d.]).
[9] Alan Marsden. 2007. Automatic Derivation of Musical Structure: A Tool for Research on Schenkerian Analysis. (Sept. 2007), 4.
[10] Alan Marsden. 2011. SOFTWARE FOR SCHENKERIAN ANALYSIS. (Aug. 2011), 4.
[11] Alan Marsden and Geraint A. Wiggins. 2008. Schenkerian reduction as search. In *Proceedings of the fourth Conference on Interdisciplinary Musicology*. Thessaloniki, Greece, 9. https://www.researchgate.net/publication/229052268_Schenkerian_reduction_as_search