12-15-2022

# Muse: A Genetic Algorithm for Musical Chord Progression Generation

Griffin Going
*Grand Valley State University*

# Muse: A Genetic Algorithm for Musical Chord Progression Generation

Griffin Going[1] and Dr. Erik Fredericks[1]

Grand Valley State University, MI, USA
goinggr@mail.gvsu.edu

**Abstract.** Foundational to our understanding and enjoyment of music is the intersection of harmony and movement. This intersection manifests as chord progressions which themselves underscore the rhythm and melody of a piece. In musical compositions, these progressions often follow a set of rules and patterns which are themselves frequently broken for the sake of novelty. In this work, we developed a genetic algorithm which learns these rules and patterns (and how to break them) from a dataset of 890 songs from various periods of the Billboard Top 100 rankings. The algorithm learned to generate increasingly valid, yet interesting chord progressions via penalties based on both conditional probabilities extracted from the aforementioned dataset and weights applied to the characteristics from which the penalty is derived. Additionally, the beginning and end of a progression may be seeded (either in totality or for a percentage of generated patterns) such that the algorithm will generate a bridging progression to connect the seeded points. To this end, the algorithm proposed chord progressions and supplied vectors of computer-aided algorithmic composition. To demonstrate the validity of the system, we present a subset of generated progressions that both conform to known musical patterns and contain interesting deviations.

## 1 Introduction

AI-generated art is an increasingly prevalent topic in the current computing landscape. Within the same domain lies computer-aided composition, wherein an artist and computer collaborate to create a piece or work. In this paper, we document an approach to computer-aided musical composition based on the grammatical evolution of chord progressions. We theorize that by providing the algorithm with a musical grammar free of specific musical rules and a data-based reference point from which to extract fitness scores for chord progressions, the algorithm will learn both common patterns within music as well as how to properly break them.

First, we briefly define grammatical evolution and provide context to the musical domain in which the subject algorithm of this work operates. Then, we offer detail on how said musical context is translated into the grammatical domain, as well as the reference point we use to evaluate the various results of grammatical evolution. With this reference point in mind, we discuss how exactly chord

progressions are evaluated based on observations made in the reference data, as well as how the fitness function manipulates the score of various chord progression characteristics to avoid convergence on a single set of outcomes. Finally, we evaluate a subset of algorithm results to conclude that the algorithm not only learned established patterns found within the McGill Billboard [3] popular music dataset, but also learned to propose deviations from these patterns that were themselves the result of established concepts in music compositions such as borrowed chords, neighboring tones, and chromaticism.

## 2 Grammatical Evolution

Grammatical evolution [4] is a form of evolutionary computation in which randomly generated collections of integers (called "chromosomes") are resolved through a tree of symbols (derived from a Backus-Naur Form grammar file) into samples composed of the terminal symbols defined in the grammar. These samples are called "phenotypes". These phenotypes are evaluated against a fitness function, providing an order to them such that we may select the top $n$ chromosomes to forward to the next generation while the remaining undergo some genetic operation and then propagate to the next generation. Through each generation, the algorithm evolves chromosomes with progressively higher-scoring attributes. An illustration of this repeating process can be seen in *Figure 1*

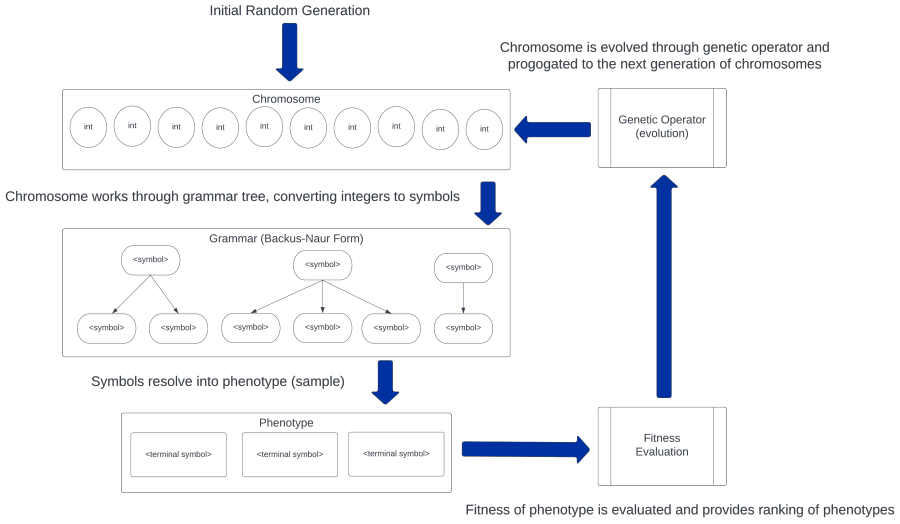



Fig. 1: Diagram illustrating the flow of a given generation of chromosomes within grammatical evolution

The specific implementation of grammatical evolution used in this work is **PonyGE2** [1], which we extend with custom configuration files to enable the following:

- Adjustable weights for various fitness-scored phenotype characteristics
- Partial seeding of phenotypes both by proportion and in totality
- Options for length-agnostic fitness (by way of converting fitness scoring to the average of phenotype characteristic groups instead of the sum of all characteristics)
- Configurable outputting of phenotypes to CSV files for analysis and external applications
- Greater control over how our custom probability-based fitness function penalizes low-probability phenotypes and awards high-probability phenotypes.

## 3 Data Types of Music

Before implementing grammatical evolution in the musical domain, we must define the relevant components of music for which we must be aware. Beginning with the simplest and moving to the most complicated unit we will encounter:

- **Note** - single unit of musical pitch (abstraction of frequency)
- **Interval** - measured distance between two notes
- **Consonance** - describes a "pleasant" harmony between two notes
- **Dissonance** - describes a "tense" harmony between two notes
- **Key** - ordered set of notes which "work" well together based on an ordered set of intervals. We may think of this as the domain of discourse within which a musical composition operates (though they may borrow from and transition to one another).
- **Chord** - a set of notes (composed of a set of stacked intervals within a given key) which emit a quality when sounded simultaneously. Selected notes are relative to (or neighbor) a key
- **Quality** - describes the set of intervals that compose a chord, such as "major" or "minor"
- **Chord Extension** - describes optional additional notes that may be added to a chord to achieve a greater complexity of sound
- **Chord Progression** - An ordered set of chords, played start to end

With these definitions, we make additional observations about how music is generally constructed:

- A chord can be broken down into three components: the note off of which it is based, the quality describing the other member notes, and the extension to which it may reach. While additional chord components exist (e.g. alterations, inversions), they are not considered in this work.
- Different chord qualities emit different genres of sound. Major chords, for example, sound "happy" while minor chords sound "sad".
- The various qualities a chord can have provide tension and relief via consonance and dissonance, creating harmonic movement. In this phenomena, chord progressions are found

Additionally, we acknowledge that while notes may be spoken of in the literal (with specific pitches assigned), we can discuss them in an abstract and relative context such that we need not consider a greater domain of discourse than necessary. To do this, we simply convert the concept of pitch from specific frequencies to roman numerals represented of pitch relative to one another. Thus, instead of pitches such as "C" and "G" (to which specific sets of auditory frequencies are mapped), we will see roman numerals such as "I" and "V" which represent the relationship between a set of frequencies instead of the frequencies themselves. An example of these (as well as their chord component breakdowns) is seen in *Figure 2*.

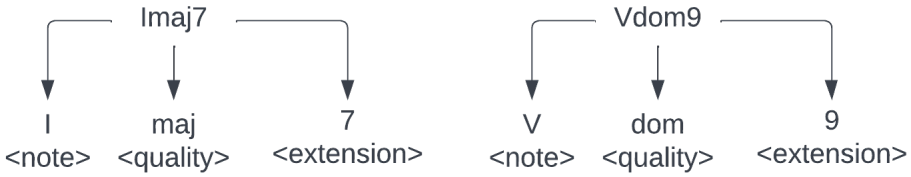



Fig. 2: Two examples of chords broken down into their composing elements

Chord progressions themselves tend to follow patterns based on the concepts of tension, relief, and harmonic movement. The dominant chord, for example, which is based off of the 5th note (roman numeral V) of a given key frequently resolves to (and thus preempts) the tonic (based off of the 1st note of the same key, roman numeral I). This pattern is likely the most important in musical composition and thus occurs with great frequency. Patterns such as this are precisely what the algorithm in this work attempts to learn, but we also desire to preserve the possibility of novelty and invention. We give the evolving algorithm no notion of these patterns such that it must learn them based on observations made of the McGill Billboard dataset.

## 4 Musical Grammatical Evolution

To perform grammatical evolution within the musical context, we adapt a subset of the aforementioned musical concepts into symbols such that they compose a grammar capable of translating the concept of a chord progression into individual chords which themselves are broken down into individual components of chords. While the possible values (i.e. terminals) of these symbols are determined based on observations in the McGill Billboard dataset, the symbols are statically defined with relationships as follows:

- **Chord Progression** - more than one chord in a sequence
- **Chord** - a note, quality, and optionally an extension
- **Note** - a roman numeral denoting the abstract note off of which a chord is based

– **Quality** - describes the set of notes included in a given chord
– **Extension** - set of optional notes which may be appended

An example of how these symbols resolve to one another (starting from the "chord" symbol) is seen in *Figure 3*.

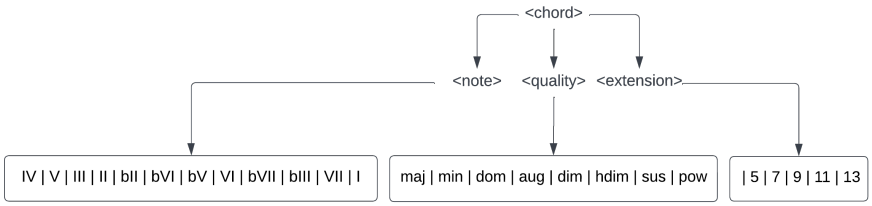



Fig. 3: An example of the "chord" symbol being broken down within a tree of grammar symbols

In addition to complete generation of progressions, we allow for the setting of both the beginning and end of a chord progression such that evolution will generate a bridging sequence between the two.

The question of population size (the number of chromosomes in a generation) is significant within the context of grammatical chord progression evolution. This significance is primarily due to the large number of possibilities present within the domain of discourse. If we assume the base case in which a chord progression is composed of 4 chords, each of which has 1 of 12 note components, 1 of 8 quality components, and 1 of 6 extension components (this varies by source dataset in our implementation), we find a total of 2,304 progressions are possible. Furthermore, this number continues to increase as the number of chords in a progression increases. Many of these progressions evaluate to a poor fitness score and thus should not be propagated to the next generation. Thus, if we use the default population size per generation of 100, we limit our domain of discourse significantly. To combat this, our population size is raised roughly equivalent to the standard variety of progressions we expect to encounter, ensuring that the population includes progressions which evaluate to better fitness scores.

Of special interest is the selection of genetic operator, which defines the function we use to evolve chromosomes. Of the primary options (selection, crossover, and mutation), we posit that crossover provides benefits not offered by the others within the context of chord progression generation. Crossover selects a number of parent chromosomes, maps $n$ points onto them, and uses these $n$ points to create a child chromosome carrying characteristics from each segment of the parent chromosomes. This encourages such musical concepts as "borrowed chords" wherein we use a chord from a parallel or nearby key, as well as other important facets of musical progressions such as chromaticism (wherein diatonic notes are briefly bridged with non-diatonic pitches). Discussion may be had on an optimal

mapping of $n$ crossover points for a given number of chords within a progression, though that is beyond the current scope of this work.

## 5 Reference Data for Probabilistic Fitness Evaluation

While the process of generating chords is data-agnostic, we require a reference from which we can evaluate progressions. A reference is not required in all implementations of grammatical evolution in the music domain, but as our fitness function evaluates various progressions based on the conditional probability that a given characteristic manifests a given way, we must have a source from which to derive said probabilities. As we use a reference for evaluation, we also use the same reference to generate the grammar file from which we derive types for generation. This creates a tight coupling between our fitness function and grammar file, offering the following benefits:

- Rigid rules need not be defined in the grammar and fitness, but can instead be naturally acquired via fitness evaluation based on observations in the reference data.
- Without the requirement of rigid rules, novelty and invention via randomness are maintained
- The universe of generational discourse is limited to what we observe in the reference data
- A reference dataset can be formulated such that patterns relegated to specific genres (e.g. rock, pop) are observed, curating an algorithm built to generate new progressions within that genre.

Worth noting is that one of the primary questions asked in this work is how well an evolutionary algorithm learns the various patterns that constitute what we consider a "good" chord progression. Were we to provide some base set of rules to which progressions must abide, we would be required to take these rules into account when evaluating progressions; we may also perhaps subsequently modify them such that generated progressions become more acceptable based on some external bias. In doing so, we interfere in the learning process and give the machine the rules of composition that we otherwise want the algorithm to learn on its own. Instead, we opt to give the algorithm as little pattern context as possible and use the reference data to provide feedback to the machine based on the frequency of various observations. This approach ultimately means that the algorithm is graded based on how well it adheres to patterns established in source material (which we validate as "good" based on its appearance in the McGill Billboard dataset [3]) without having prior knowledge of these patterns.

The McGill Billboard Project [3] is custodian to a dataset of 890 songs complete with metadata, timestamps, and chord progressions separated into various sections of a song (e.g. intro, verse, chorus, etc). To adapt this data to our purpose, we perform the following steps:

1. Parse the string representations of chords into a map of sections (such that each section of each song is individually addressable if we wish to limit our universe of discourse to specific songs and/or sections).
2. Collapse groups of similar sections (e.g. multiple choruses) into their greatest common components (preventing longer songs with a greater number of repeated sections from more heavily weighting probabilities than those of more standard length)
3. Collapse the chord progressions of sections into their smallest repeated patterns
4. Evaluate various predefined characteristics of each progression, documenting the conditional probability that the observed characteristic occurs in a map of all probabilities relevant to our probability-based fitness function.
5. Generate a grammar file from the keys found in the probability map

A visualization of this approach may be seen in *Figure 4*.

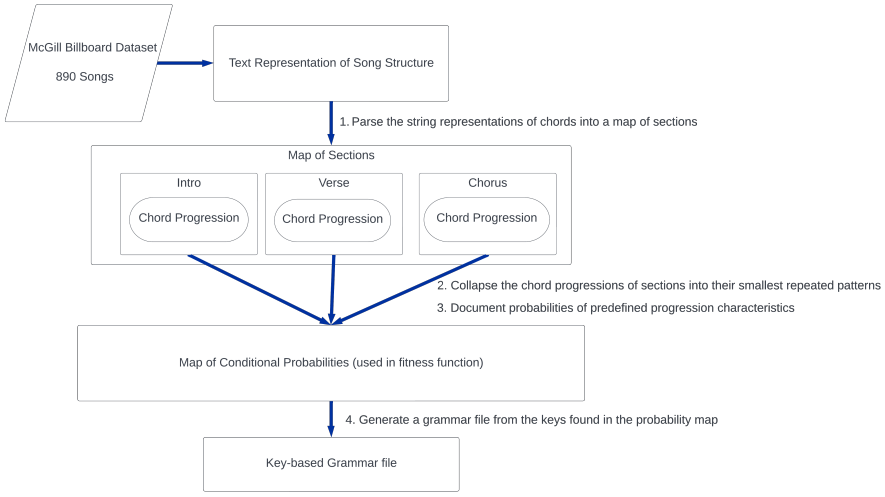



Fig. 4: Preprocessing the McGill Billboard dataset

## 6 Fitness & Progression Evaluation

As previously stated, a primary goal of this work is to observe an evolutionary algorithm learning the various patterns that compose chord progressions without explicitly providing those patterns as rules which must be followed. This becomes more difficult when presented with the problem of evaluating a given progression; namely, if we cannot explicitly state what "good" is, how can we provide a relative order for a generated set of progressions? To achieve these, we use the map of conditional probabilities extracted from the McGill Billboard dataset [3] which describe the prevalence of various chord progression characteristics. Thus,

a given chord progression is scored based on the following characteristics (shown in *Figure 5* tied to their relevant chord components):

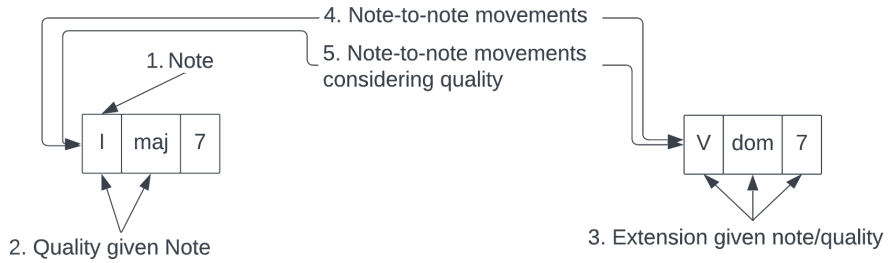



Fig. 5: Diagram of two isolated chords from a progression with scored characteristics labelled

Additionally, we score progressions on both the starting and penultimate/ending chord(s).

This approach allows us to score chord progressions based on observations within the source dataset instead of concrete rules to which they must subscribe, allowing observable patterns to emerge while still containing levels of invention due to the inherent randomness of symbol resolution and generation.

We then define fitness as follows:

$$\sum_{i=1}^{n}(1-x_i)^m$$

wherein:

- $\boldsymbol{n}$ is the number of chord characteristics for a given progression
- $\boldsymbol{x}$ is the conditional probability that the observed manifestation of a given characteristic occurs
- $\boldsymbol{m}$ is a configurable exponent which modifies the penalty curve of the function for each observed characteristic

An alternative for length-agnostic fitness:

$$\sum_{j=1}^{c}\left(\frac{\sum_{i=1}^{n}(1-x_i)^m}{n}\right)$$

wherein:

- $\boldsymbol{n}$ is the number of chord characteristics for a given progression for a specific characteristic

- $\boldsymbol{x}$ is the conditional probability that the observed manifestation of a given characteristic occurs
- $\boldsymbol{m}$ is a configurable exponent which modifies the penalty curve of the function for each observed characteristic
- $\boldsymbol{c}$ is the number of defined characteristics upon which fitness is evaluated. We define 8 such characteristics.

The effect of m is most easily observable for a single characteristic and can be seen in *Figure 6*. Recall that PonyGE2 [1] seeks to minimize fitness scores, and we therefore seek to assign a high fitness value to "poor" progressions and a low value to "good" progressions.

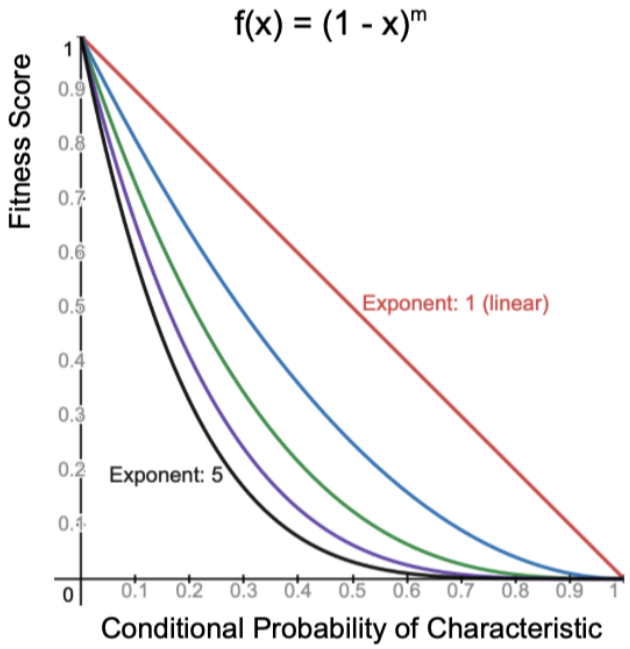


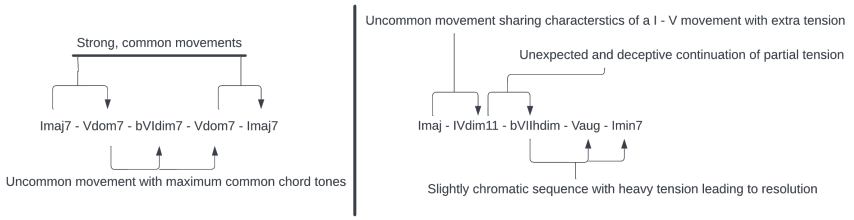

Fig. 6: Graph of the fitness function input and outputs for any arbitrary progression characteristic based on the conditional probability for a given characteristic occurrence

We observe that increasing the exponent $m$ continues to penalizes characteristics that manifest in very unlikely forms while lessening penalties on manifestations that are only somewhat unlikely. Additionally, the curve when greater probabilities are encountered is flattened such that we do not converge on the single most likely outcome, but rather that outcomes above a certain likelihood are roughly and increasingly equivalent. Through this method we prevent converging on the single "best" progression based on the source data and instead converge on a set of high-scoring progression characteristics above a configurable threshold.

## 7 Results

The progression of best fitness is, without fail, always some instantiation of the aforementioned I - V - I progression and the related I - IV - I progression. This is unsurprising as these progressions are among the most standard cadences in music. Such outputs as these are an indicator that the algorithm is in fact learning to create progressions that subscribe to existing patterns.

Of greater interest are the progressions that the algorithm proposes of less traditional movements as a result of deviation from the learned patterns. Two examples can be analyzed as follows:




We conclude that not only did the algorithm learn established patterns found within the McGill Billboard popular music dataset [3], but also that the algorithm learned to propose deviations from these patterns that were themselves the result of established concepts in music compositions such as borrowed chords, neighboring tones, and chromaticism.

## 8 Further Work

This work provides a platform from which additional experiments may be conducted and further developments made. These include but are not limited to:

- Expansion of considered chord components such as chord alterations and inversions
- Experimentation with additional genres and datasets
- Dynamic chord characteristic weight optimization based on desired progression outcomes
- Chord "search", wherein generated progressions are selectively reported based on user-specified criteria
- Domain-specific genetic operators, such as $N$-point crossover wherein pairs of points are limited to placements enclosing complete chords

The code corresponding to this work is available on GitHub [2]

## References

[1] M. Fenton et al. *PonyGE2: Grammatical Evolution in Python. arXiv preprint, arXiv:1703.08535*. 2017. URL: https://github.com/PonyGE/PonyGE2.

[2] G. Going. *Muse*. 2022. URL: https://github.com/GriffinGoing/Muse.

[3] Jonathan Wild John Ashley Burgoyne and Ichiro Fujinaga. "An Expert Ground Truth Set for Audio Chord Recognition and Music Analysis". In: *Proceedings of the 12th International Society for Music Information Retrieval Conference*. Ed. by Anssi Klapuri and Colby Leider. Miami, FL, 2011, pp. 633–38.

[4] Conor Ryan, J. J. Collins, and Michael O'Neill. "Grammatical Evolution: Evolving Programs for an Arbitrary Language". In: *Proceedings of the First European Workshop on Genetic Programming*. Ed. by Wolfgang Banzhaf et al. Vol. 1391. LNCS. Paris: Springer-Verlag, 14-15 4 1998, pp. 83–96. ISBN: 3-540-64360-5. DOI: doi:10.1007/BFb0055930. URL: http://citeseer.ist.psu.edu/ryan98grammatical.html.