

Automatic Music Transcription Using Genetic Algorithms and Electronic Synthesis

David Lu!!

April 25, 2006

Abstract

Polyphonic music transcription is a computationally difficult problem that is still widely regarded as unsolved. This paper describes one method for the problem of pitch recognition with multiple instruments. A genetic algorithm produces generations of possible transcriptions that are evaluated for fitness by comparing the transcription's generated audio to the original sound file. By sharing and modifying the genetic material of thousand of individual possible transcriptions, ultimately, the algorithm arrives at an accurate transcription for the original audio, or at least something that produces a similar wave form.

1 Background

1.1 Definitions

Automatic music transcription is the process in which a program takes a recording as input and outputs what notes are playing at what times. Here, we are transcribing *polyphonic music*, which is music in which there is more than one sound occurring at the same time, with multiple notes on one instrument (like a piano) or single notes on multiple instruments. An *audio representation* of music is a digital recording, from which the frequencies heard at any particular time can be determined. This can take the form of an mp3, CD recording or any other representation of the physical sound intensities. A *musical representation* or *transcription* is more like sheet music; it contains specific musical information for what notes are played when, but it does not specify exactly what the result will sound like. The musical representation on computers usually takes the form of a MIDI file. Figure 1 below shows different ways the representations are visually depicted. Finally, when we refer to a *song*, we mean the entire selection that is being transcribed.

1.2 Physics of Music Basics

We sense sound by noticing minute differences in the pressure of the air outside of our ears. It is these intensities that can be reproduced by speakers, or recorded from a microphone.

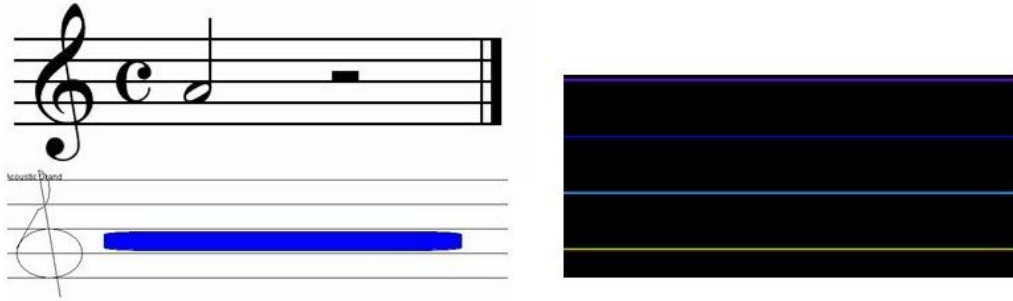


Figure 1: Visual Representation - This figure shows the different ways the music is represented in the program and this report. In the upper left, we have the standard musical representation. Below that is a more precise notation, which shows precisely when the note starts and stops. While not as readable for performance, it is much better at displaying the rhythm of complex overlapping sets of notes. On the right is the representation of the audio, or rather, of the frequency spectrum. Similar to the other diagrams, time is represented by the horizontal axis, while lines closer to the top represent higher notes.

The patterns produced by these intensities are indicative of the three basic properties that notes have: pitch, timbre and dynamic.

The patterns have a cyclical nature, i.e. they oscillate at a certain rate. We can determine the signal’s basic frequency, which leads us to the concept of pitch. Pitch dictates how “high” or “low” the note sounds to us. Secondly, there is the timbre of the note, which is what the note sounds like. Differences in instrument are the most usual source for different timbres. What this means physically is that in addition to the basic pitch (called the fundamental frequency), there are sounds of other frequencies as well. The most intense other frequencies happen on multiples of the fundamental frequency, also known as harmonics. If, say, we were playing an A, which is 440 hertz (cycles/second), then the first harmonic would be twice that, at 880 hertz, and the second would be at 1320 hertz, three times the fundamental.

Finally, there is the dynamic, or how loud the instrument is playing. While this is a fundamental component of music, for purposes of this paper, changes in dynamic are ignored. This includes attacks and decay, meaning that although notes usually start out quite loud, then taper off in volume, the dynamic of the music in this scope remains constant.

1.3 Prior Work

In an article entitled “Automatic Music Transcription as We Know it Today,” where today refers to 2004, Anssi P. Klapuri discusses the state of the music transcription problem. He states, “[d]espite the number of attempts to solve the problem, a practically applicable, general-purpose transcription system does not exist at the present time.”[1] Different researchers have used different approaches to transcribe as many as three different simultaneous sounds.

Alain de Cheveigné and Hideki Kawahara were able to estimate pitches with a *degenerative* process, by subtracting one note estimate from the frequency spectrum at a time. However,

this model is limited by the fact that small perturbations in the audio will result in imperfect subtractions, leading to larger problems with the transcription as a whole.[2]

Manuel Davy and Simon J. Godsill used a probabilistic/statistical Bayesian network and variable-weight sound model as a heuristic to transverse the large parameter space of music transcription. They were able to transcribe three simultaneous notes, but no more.[3]

Masataka Goto created a method that was able to transcribe the melody and the bass line among any number of instruments. By guessing every possible fundamental frequency, Goto was able to figure out the transcription approximately 80% of the time.[4]

There are dozens of other approaches. Here at the University of Rochester, Gordana Velikic is attempting to transcribe music using not only cues from the frequency, but the phase of each individual wave.[5] Many approaches to the problem of music transcription, including hers, are firmly based in signal processing, i.e. looking at the frequency spectra of the sounds, making mathematical judgments and proceeding from there. This may not be the best method.

2 The Algorithm

2.1 The New Approach

The problem with the current methods is that they attempt to conceptualize a process that is unknown. There is no set way to start with the audio and end up with the musical representation. Like vision or inverse kinematics, this is a difficult unsolved problem. The methods described above make strong attempts, but in the end, they are still trying to quantify a process that is not only computationally unknown, but also only seen in the real world among skilled musicians. Klapuri likens the process to “reverse-engineering the ‘source code’ of a music signal.”[1] The fact is, this process will never be exactly understood.

However, the *forward* engineering process is known. Going from a musical representation to an audio representation is what MIDI cards and synthesizers perform all the time. Given a sufficient instrument model, it is easy to generate what any set of notes sounds like. Hence, it makes sense to use this advantage. We move the computational complexity away from finding the correct model for the complex signals. Instead, the complexity moves to the real problem, figuring out who is playing which notes.

Our method uses a “gen-gen” system that combines a *genetic* algorithm and music *generation* system. In the conclusion of Klapuri’s summary of transcription, he stresses the need for both a method for analyzing the music and a means of parameter optimization. The gen-gen system fits those precise requirements. The generation system gives the algorithm the material to analyze, while the genetic algorithm returns improved hypotheses. Figure 2 shows this program flow.

This approach is beneficial because it is not limited by any particular harmonic model. A more traditional approach might find the highest intensity frequency and automatically assume that it is the fundamental frequency. Or, it might confuse two instruments that are playing the same note for one, since their overtones overlap so much. The fact of the

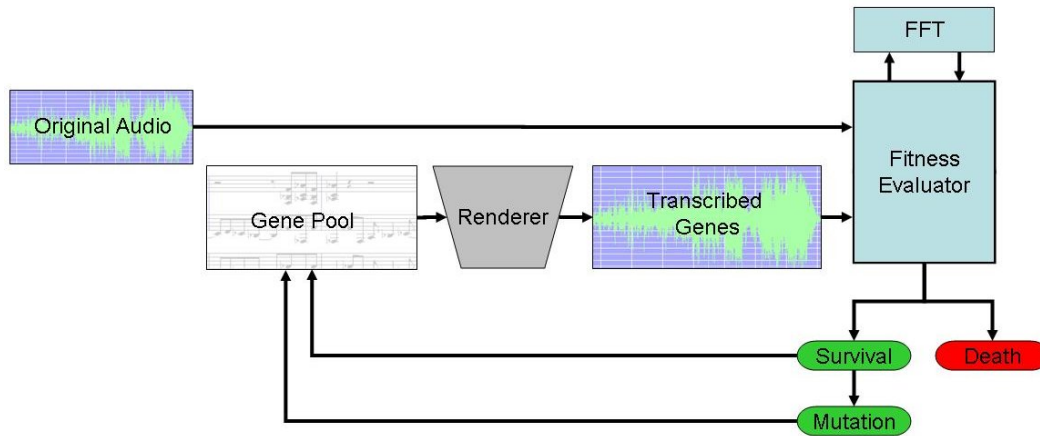


Figure 2: Program Flow - The program starts with the original audio in an audio representation. The genetic algorithm starts by creating the gene pool, containing a selection of possible transcriptions. Each transcription is then passed to the Renderer, which transforms the musical representation into the audio representation using the generation system. Each has its fitness evaluated by using a Fast Fourier Transform on both the original audio and the new possible transcription’s audio to create their power spectrums, which are then compared to each other.

matter is that polyphonic music creates a complex frequency lattice that is computationally infeasible to deconstruct. However, with the gen-gen system, this lattice will not need to be deconstructed, it will rather be reconstructed. It is our claim that by mimicking the process with which the audio was originally constructed, the transcriptions produced can get much closer to an ideal transcription.

2.2 Genetic Algorithms

Genetic algorithms are one of many tools used to explore large search spaces. The idea is to simulate the evolutionary process that species undergo in nature. Genetic algorithms emulate survival of the fittest, with the fittest individual at the end of our simulation being our solution in the search space. A collection of “individuals are created in the Gene Pool. The less-fit individuals die away and the more-fit individuals live and go on to reproduce. Their reproduction creates slightly altered versions of themselves, using some sort of mutation of their genetic material. These mutations are based on both asexual reproduction, where the genetic material from just one individual is changed, and sexual reproduction, where the genetic material is combined from two different individuals. The mutations can improve or hurt the individuals, however, the less fit individuals will die off, leaving the more fit ones to reproduce and create even better versions of themselves in their offspring. This process results in ever-increasingly fit genetic material, which for us, means better and better transcriptions of the music.

The twist however is that these individuals are not real individuals. They are merely data structures which simulate the above process. In order to figure out whether each individual

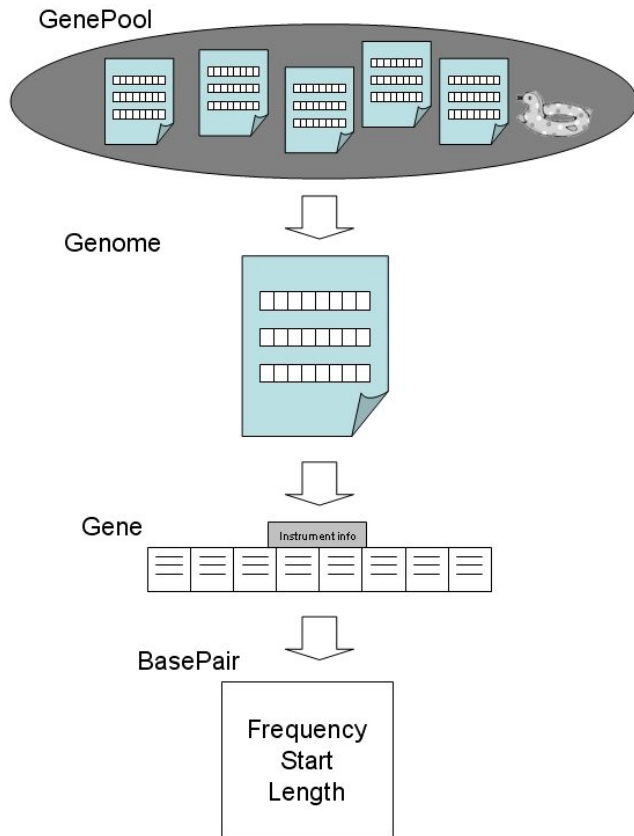


Figure 3: The hierarchical structure of the genetic material.

survives or not, since we cannot watch it live or die, it is evaluated by applying a fitness function. Furthermore, the genetic material is not actually DNA, but rather a pattern of data that simulates DNA.[6]

2.2.1 Internal Representation

In most genetic algorithms, the genetic material is stored as a bit string, which is then interpreted for the specific problem. This allows the genetic algorithm to perform basic operations on the genes in a way that most closely simulate the biological mutations. However, in a problem as complicated as musical transcription, it was deemed more fitting to create a genetic representation in which salient features such as pitch and timing, would be separated so that one type of information could not be stored as the other. Hence, instead of a bit string on which to perform operations, the genes are composed of a hierarchical structure which is not dissimilar to the internal representation of a MIDI file. This hierarchical structure can be seen in Figure 3. Typically, there are 200 genomes in the Gene Pool, with each genome having a couple genes and each gene having a dozen or so base pairs. However, each of these numbers is highly variable (except for Pool size) due to the massive variations in music.

As with most genetic algorithms, there is a Gene Pool that contains each of the individuals, or in this case, the possible transcriptions, which we call “Genomes”. Each genome is divided into several “Genes,” analogous to a MIDI file’s tracks. Each of these represents the part played by one type of instrument. This could mean one clarinet (which only plays one note at a time), or a piano or ensemble of clarinets (which plays multiple notes at one point). The structure is divided up even further into BasePairs, each representing one note. The BasePair has only three features: frequency, start time and length. Frequency is limited to the 128 different notes available in MIDI, from a low C, frequency 8.18 Hz, to a high G, frequency 12543.88 Hz.

2.2.2 Time Division

The placement of notes in the Genome is based on absolute, not musical timing. We could define each note in the way that sheet music does, with each note being some fraction of a beat. However, this sort of information makes it difficult to account for tempo changes and notes of irregular length.

Instead, we store all the temporal data, i.e. start time and length, is stored internally in a fashion similar to MIDI files as well. Time increments are assigned an arbitrary integer value called “ticks,” where beat 1 will be at tick 0, and beat 2 might be at tick 1024. This allows for the division of the intervening time to be assigned with greater freedom than if it were done with the smaller integer beat values.

However, these are only for the internal representation. The smallest functional time unit is the “time-slice.” This is defined for each transcription as the smallest interval of time for which we want discrete note information. For example, if we want to transcribe a set of eighth notes, the eighth note should be the time slice.

When the fitness is calculated, it does so by computing a separate FFT for each time slice. This way we can track changes in pitch as they happen. Also, for simplicity, generated time values (i.e. where notes start and end) are rounded to coincide with the starts of the time slices.

2.2.3 Essential Problem

The essential problem with music transcription is that it is a search in a large multi-dimensional space. There are a finite number of possibilities for each moment, meaning that for a piece of any particular length, there are a finite number of possible transcriptions for it. This is what allows us to search in the first place. However, it is a huge search space. Considering that there are 128 different notes, plus the option of not playing, if the ensemble being transcribed has two different instruments, each capable of only playing one note at a time, there are over sixteen thousand different configurations for one time slice. Thus, when you combine just three notes consecutively, there are over 4 trillion combinations.¹ Clearly, iterating through all possible solutions is impractical.

¹ $129^2 = 16,641$ and $129^3 = 4,608,273,662,721$

However, more than likely, the music to be transcribed is not a random collection of notes in time. Rather, it will be some sort of coherent piece, which follows given patterns and has specific tendencies. Some notes will be seen more than others. Musical material repeats itself all the time, sometimes with minor variations. We use the mutations and structure of the genetic algorithm to exploit these tendencies, allowing for a more refined search for the transcription.

2.2.4 Selection

Each generation, after the fitness is evaluated, some individuals will be less fit than the others, so in keeping with the philosophy of the genetic algorithm, some must ‘die’ and some must go on to reproduce. (The fitness function is discussed in Section 2.3.2.) The selection of individuals for mutation is strictly survival of the fittest. There is no chance involved here. The genetic algorithm simply kills the bottom third of the population and for each individual in the top third of the population, creates a mutation and adds it to the gene pool. This keeps the population constant, with the new population consisting of one third old individuals who just reproduced, one third old individuals who did not reproduce, and one third new individuals. Since there is no chance involved in the selection and genes that reproduce do not die (during the generations where they reproduce), there is no need to keep the extra genes around. We could select a smaller fraction of the population for death and reproduction, but the other genes would never be used.

2.2.5 Mutations

Because of the added complexity of the internal representation, the mutations have been tailored specifically to fit the structure of the genes. They need to take into account the variable gene length and the hierarchical structure. There are six different mutations, detailed in Table 1.

Note that crossover is not among the mutations listed, despite it traditionally being the most common mutation. Crossover entails swapping genetic material between two individuals, giving some of the material from A to B, while removing it from A. However, in this problem, the genetic material found inside high-fitness individuals is good enough such that most of the material is at least partially correct. Thus, removing it to add to another individual is actually detrimental to the donating individual, and hence it is more efficient to copy the material, as done in the assimilate mutation. Further, by tracking that material, it was found that the assimilate function vastly outperforms crossover. Thus, it is not included.

2.2.6 Mutation Selection

Roulette selection is used to determine which mutation to apply to each selected individual. The idea is that each mutation has a section of a roulette wheel, and if a random spot on the wheel is picked, that mutation is selected. If all the sections of the wheel were equal, then each mutation would be picked an equal number of times. Mutations do not produce the same results, so it makes more sense to give them different weights and have some mutations

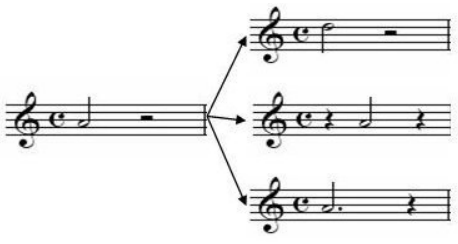
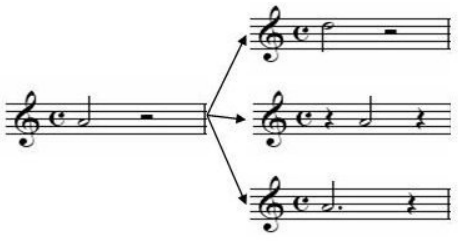
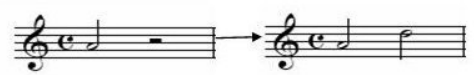
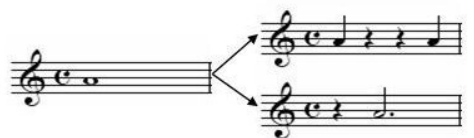
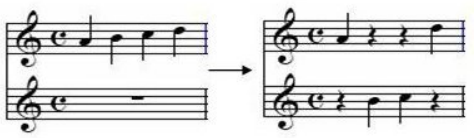
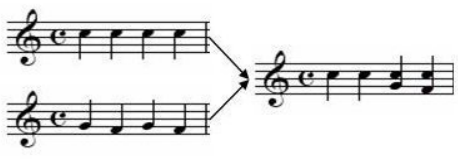
Name	Description	Illustration
Irradiate	Randomly changes a feature of a BasePair. This can result in an alteration of one of three things: its pitch, its start time or change its end time. This is the most basic of the mutations and has the highest randomness.	
Nudge	Works exactly the same as Irradiate, but only changes each feature incrementally, by the smallest amount possible. Hence, the pitch is changed by one semitone, or, the start or end time are changed by one time slice, i.e. the smallest amount that will not be rounded off.	
Lengthen	Adds another BasePair to a Gene. This serves as the primary asexual way of increasing the size of the transcription, i.e. adding more notes.	
Split	Inserts silence to a BasePair. The silence can span the entire note, thus removing it, or it could appear at the ends, thus shortening the note, or it could appear in the middle, splitting the note into two.	
Reclassify	Moves a section of one Gene to another, possibly new, Gene. This allows multiple instruments to be introduced and for information to be shared between different Genes within the same individual.	
Assimilate	Takes a section of a Gene from one individual and copies it to another genome. This is the only sexual mutation, since it takes material from two different individuals. Along with lengthen, assimilate increases the size of the gene.	

Table 1: The specialized mutations used by the genetic algorithm

be performed more often than others.[6] However, taking the idea further, not only are some mutations more useful than others, but some mutations are useful than others at different times. For example, small incremental changes are often most useful towards the end of a transcription to fix minor details, but at the beginning, they are ineffective at quickly finding the bulk of the information. Thus, by changing the proportion of the roulette wheel that the mutation occupies over time, we have an optimizing method that allows for quicker transcriptions.

2.2.7 Termination Condition

There are two conditions with which the genetic algorithm can be terminated. First, if the fitness, which is always in the range $[0,1]$, of the best individual exceeds 0.99, then the program assumes we have a correct answer and terminates. However, we must also consider the case where the fitness does not converge to 1. So we define a constant to dictate how long the program should wait before giving up. For this, we consider the number of generations that have gone by since the last improvement in fitness. If that number accounts for more than half of the number of total generations, and that number is greater than 200, then the program has decided it cannot make any more progress, and gives up.

2.3 Music Generation

In order to compare the original audio and the candidate transcriptions, we need two things: a common information representation and a way to transform the transcriptions into the audio format. The information format used to store audio is sampled audio, represented internally as a one dimensional byte array where the bytes represent the sound intensity at the particular time. This is roughly equivalent to an uncompressed WAV file.

2.3.1 Instrument Modeling

To transform the genetic material into real audio, we need to know what each instrument should sound like. Ideally, what we would like to do is record actual instruments playing each possible transcription and compare that to the original recording. However, musicians' wages being what they are, recording each of the thousands of possible transcriptions seems implausible. Furthermore, the variability in the sound produced by real musicians would lead to too much error.

Instead, we use simple instrument models. In this particular implementation, we use well-known mathematical instrument models and several models created through additive synthesis. The mathematical models are the square, saw tooth and triangle waves, universally defined, with harmonics going up to the Roche limit (i.e. the highest meaningful frequency allowed by the sampling rate). The additive models are created by analyzing real instrument sounds and replicating the intensity of their harmonics. This creates harmonically similar sounds, but when played, are not likely to be confused for the real thing. This particular approach is by no means perfect, but suits the purpose of testing the philosophy behind this approach.

For each note, an array containing the intensity of the wave is created, and then added into the sampled sound of the entire song. The voices are combined by simply adding up the magnitudes of all the sounds. Each of the note arrays is cached, and often retrieved later for speed increases.

2.3.2 Fitness Function

Once we have the original and candidate transcriptions in the same format, we can compare them to evaluate the fitness of each transcription. However, a straight comparison which measures the difference between the samples would result in similar sounds being rejected because of minute differences in small factors like phase. Instead, we chose to work in the frequency domain. The power spectrum, for instance, throws away phase so that differences in the phase of sine waves making up the signal are Hence, in order to most effectively compare two sounds, a fast Fourier transform (FFT for short) is performed on each sound, allowing us to compare the magnitudes of all the frequencies.

Performing such a transform leads to the phenomenon of “leakage”, stemming from the differences between the measured frequencies and the size of the sample. To ensure there are no artifacts of unwanted frequencies, we use a Hann windowing function, $w(n) = 0.5(1 - \cos(\frac{2\pi n}{N-1}))$ which in tests, reduced the amount of leakage in the frequency spectra.

Specifically, the fitness for any particular gene is the sum of the difference between each frequency in each time slice of the song, squared or,

$$Fitness = \frac{1 - \sum_{t=0}^{tmax} \sum_{f=fmin}^{Rochelim} (O(t, f) - X(t, f))^2}{\sigma}$$

where $O(t, f)$ is the magnitude of frequency f at time t in the input audio, and $X(t, f)$ is the same for the possible transcription. σ is a scaling factor equivalent to the first worst transcription. The scaling factor puts most fitnesses in the range $[0,1]$, with 1 being a perfect match. Fitness is computed from 0 to $tmax$, traversing all time from the beginning to the end, and from $fmin$ to the *Rochelim*, which are the smallest and largest frequencies possible.

3 Results

3.1 Testing Methodologies

3.1.1 Process

For testing the effectiveness of this algorithm, we needed test cases for which we could easily check the accuracy of the transcription. Rather than deal with the unexpected features presented in actual recordings, where the sound truth is questionable, synthesized input, whose ground truth transcription is known, presents more controlled and more flexible input than real recordings. In fact, for the initial testing of the gen-gen system, it was easiest to synthesize input using the rendering engine described above. This means it is possible to get a perfect transcription.

The synthetic input for testing is gathered by first inputting a musical representation, which can take the form of either a MIDI file or certain parameters which can be used to build a musical representation. That representation is rendered into an audio representation, which is then used as the input to the transcriber, i.e. the original audio seen in Figure 2.

Once the transcriber has terminated, the best transcription is then compared to the original *musical* representation to check exactly how accurate the transcriber was compared to what we know the input was. This is done by comparing the intersection of sets. For any given time slice, let O be the set of notes playing on particular instruments in the original input and T be the set of notes for the transcription. Also, let M be the set of notes that are being played by the wrong instrument. Then we define the *exact rating* for that time slice as follows.

$$ExactRating = \frac{|O \cap T| + .5 |M|}{|O \cup T|}$$

These numbers are then combined for each time slice to get an exact rating in the range $[0,1]$ for the entire transcription, with 1 being a perfect match. Using this measure, we will be able to figure out precisely how accurate the transcriptions are. ²

3.1.2 Difficulty Scale

There are three different factors to consider when deciding what makes music hard to transcribe. First, we have the number of simultaneous notes per instrument. Then, there is the length of the piece, measured in time slices. This is the controlling variable as to how long the algorithm runs, since the longer the piece is, the more FFTs that must be computed. The other two factors are the number of instruments and the number of simultaneous notes. The examples listed below will have their difficulty rated by the triplet (x, y, z) , where x is the number of notes per instrument, y is the number of instruments and z is the length. Triplet ratings with higher numbers are harder than those with lower numbers.

These factors can also be used as parameters for generating a musical representation

3.2 Capabilities

The gen-gen system is able to transcribe many different pieces accurately. The process works as one would expect, with the algorithm slowly filling in different gaps in the transcription, as shown in Figure 4.

On the most basic level, the transcriber can robustly transcribe monophonic songs. "Row Row Row Your Boat" for example, has a rating of (1,1,48) and is transcribed in 233 generations. This demonstrates the gen-gen system's ability to actually produce concrete transcriptions. From here, we move on to tests that are more complex.

For more complete testing, musical representations were generated to fill up a swath of the triplet parameter space, ranging from (1,1,1) to (3,5,5), filling in all values in between. Each triplet value generated ten musical representations filled with random notes on the

²The reason why this is measure is not used for the actual fitness function is that in the actual transcription process, we will not have access to the original music representation

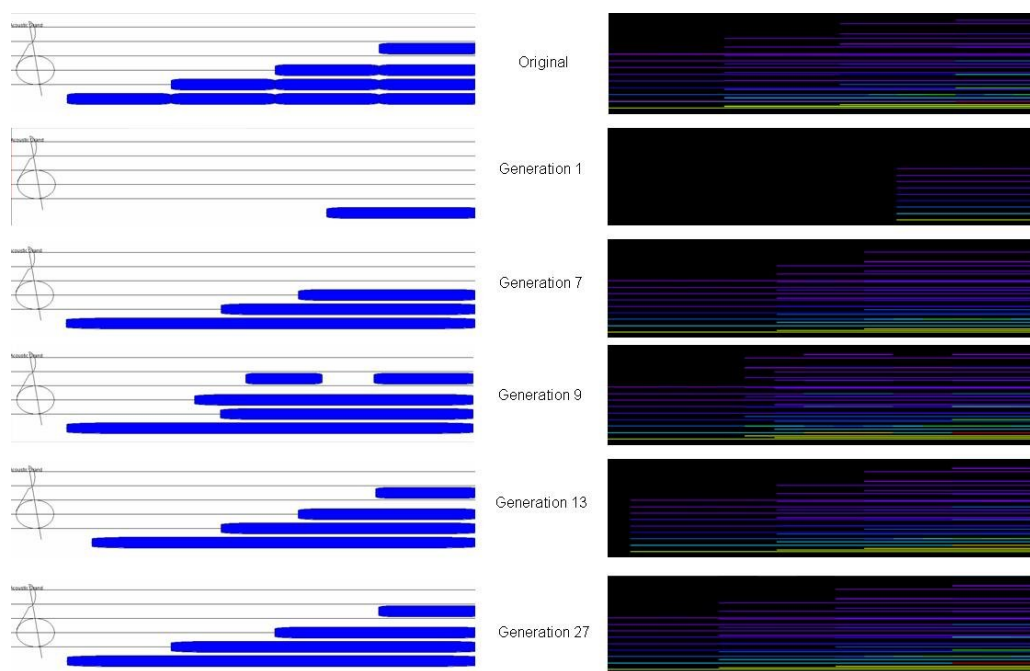


Figure 4: The progress made by the algorithm on a simple building chord (4,1,16). On the left is the musical representation, showing which notes are being played. On the right is the audio representation, showing the frequency spectra.

treble and bass clef (ranging from a low G to the high F approximately three octaves above it. These generated songs were then run through the transcriber.

The results were surprisingly good. For most of the parameter triplets, the transcriber was able to get at least one transcription that had an exact rating of 1.0. In fact, only 14 of the 75 never got a perfect exact rating. This is impressive in sheer number of notes transcribed. The most notes transcribed resulting in a perfect exact rating was 32, using the triplet (2,4,4). However, not only that, but if we consider the (x,y,1) parameter section, i.e. the section with only one time slice, we were able to accurately transcribe (3,5,1) in all trials. That means we transcribed a whopping fifteen notes on five different instruments per time slice. When compared to the three possible notes transcribed by prior implementations of transcribers, the number is phenomenal. The sheer possibility of accurately transcribing that many notes at one time is exciting. Consider the fact that for this parameter triplet, there are 4×10^{31} different possible transcriptions.³

Figure 5 shows the average exact fitness graphed over the total number of notes in the original song. The graph shows a clear linear regression of exact fitness as the number of notes increases. Theoretically, if it continued along that path, it would max out at 173 notes. However, that is unlikely.

When there are fewer notes to transcribe, the results are much better. The question then becomes why the algorithm is not run individually on each time slice, thus reducing the number of notes that need to be kept track of. This issue shall be investigated further,

³ $(129 \times 128 \times 127)^5$

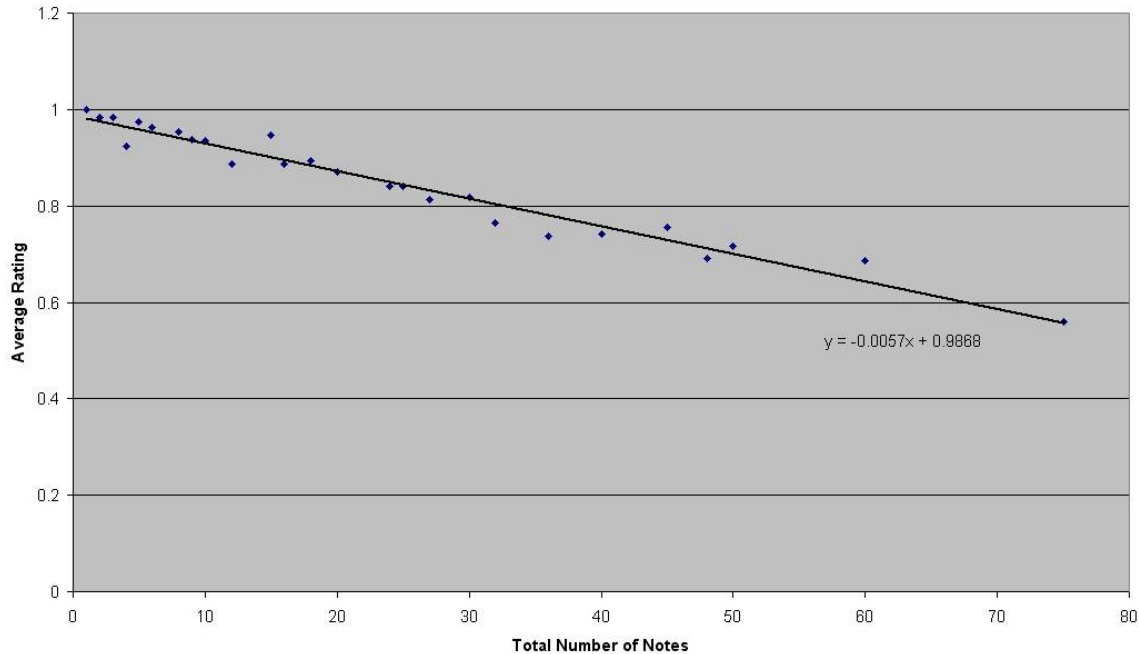


Figure 5: Average Exact Rating vs. Number of Notes

but our claim is that by examining the song as a whole, we are able to take advantage of information we have already discovered in other sections, thus making the transcription happen much quicker.

Figure ?? which shows a quadratic relation between that the number of notes and the number of generations it takes to terminate.

4 Future Work and Enhancements

4.1 General Improvements

The algorithm fails in some cases for two reasons. Occasionally it will still become stuck in local maxima. One solution to this which was not implemented was routinely checking the gene pool for similar genes.

The algorithm also sometimes fails when there are two notes within a half step of each other, where only one of the notes is transcribed. The best way to solve this problem is unclear.

Further improvements could be made by allowing for different dynamics to be reflected in transcriptions. As it is now, each note is considered the same volume. This extension could happen by adding another feature to the BasePair structure. However, this would increase the difficulty.

Also, it is imperative that the system be extended to work on real audio recordings and with more instrument models.

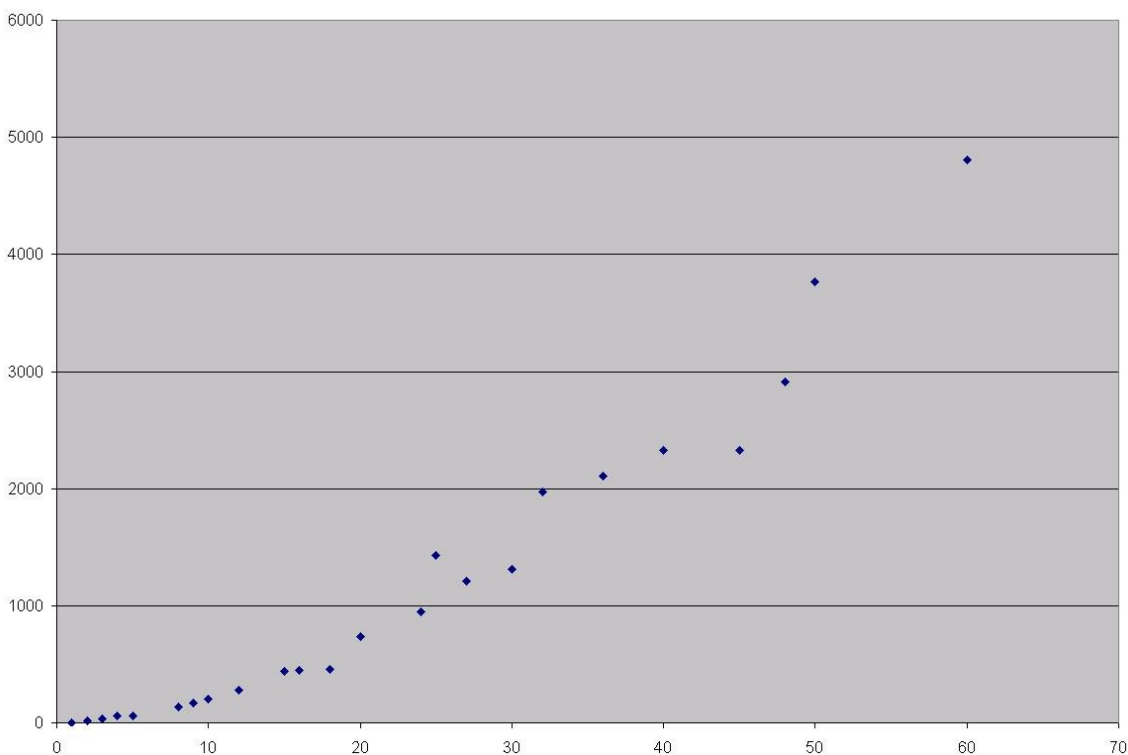


Figure 6: Average Number of Generations until Termination vs. Number of Notes

4.2 Complexity-Based Locality

Another problem with a large search space such as this one is that it inevitably has many local maxima. One way to avoid being stuck at a local maximum is to ensure diversity in the gene pool. This allows useful mutations to emerge that can help move away from the local max. However, in this problem, fitness is directly dependent on number of correct notes, which means less complex genomes that develop an advantageous mutation will have relatively low fitness since they do not have nearly as many notes as the other genes. Hence, it is hard for new ideas to mutate because, despite the fact that they have good possibly unique genetic material, their advantages are dwarfed by the more complex genes that have much higher fitness based solely on their greater number of notes.

Thus, in order to ensure the generation of new ideas is not stifled, complexity-based locality is introduced. Conor Ryan[7] discusses a generic model of locality which he calls “Isolation by Distance” which carves out niches for the different populations, each with a separate gene pool. The individuals reproduce on their own for the most part, but occasionally emigrate to another place.

In the complexity-based model, the different niches are not considered equal. Each has individuals of a different complexity. The emigration occurs when an individual is produced that is a higher complexity than the niche it is in. When applied to the music transcription domain, the complexity-based locality would allow smaller features such as measures to develop in one niche, which would then be able to more complex track or song without as

much of a fear of becoming stuck with a suboptimal genome.

4.3 Diagnostics

In traditional genetic algorithms, the fitness is computed only for the individual as a whole. However, in this particular problem, the fitnesses of the smaller units can be computed as well, because the fitness of the individual is just an average of the fitnesses for each time slice. In this way, it is not only possible to access fitness for the individual as a whole, but also as a function of time. The fitness for each time slice can be then reflected back onto the genetic material that produced it, allowing the most problematic areas to be addressed.

It is possible to examine each individual in terms of where the lowest fitness is. This is done by calculating the average fitness for each time slice during which a note is played. Once that is done, the algorithm can calculate where, within the individual, the most work is needed, and then make a mutation there.

4.4 Advanced Fitness Function

Currently, the fitness function works in such a way that exact matches are rewarded highly. This is what we expect, however, it does not reward close matches. Clearly, an F# is closer to an F than say a Bb. However, under this system, both the F# and the Bb count for nothing. Similarly, a C on beat 1 does not match up with a C on beat 2. Accounting for these cases would require a more complex fitness function that checked the values of the neighboring times and frequencies in a small window.

5 Conclusion

5.1 Summary

Using the new gen-gen system, we were able to correctly analyze songs with much greater complexity than the existing systems. Not only can the transcriber correctly identify the notes being played but the instruments which produced them. By avoiding the traditional signals processing approach, we were able to uncover an elegant way of transcribing music, allowing a computer to come that much closer to the level of sophistication that real musicians have.

5.2 Acknowledgements

I would like to thank Professor Mitsu Ogihara and Professor Chris Brown for their wealth of knowledge and continuous encouragement throughout the year, and to Professors Mark Bocko, David Hedlam and Ted Pawlicki for their support as well.

References

- [1] Anssi P. Klapuri. *Automatic Music Transcription as We Know it Today* Journal of New Music Research. 2004, Vol. 33, No. 3, pp.269-282
- [2] Alain de Cheveigné, Hideki Kawahara. *Multiple period estimation and pitch perception model*, Speech Communication 27, 175-185
- [3] Manuel Davy and Simon J. Godsill. *Bayesian Harmonic Models for Musical Signal Analysis* Bayesian Statistics VII, Oxford University Press, 2003.
- [4] Masataka Goto. *A Real-time Music-scene-description System: Predominant-F0 Estimation for Detecting Melody and Bass Lines in Real-world Audio Signals*, Speech Communication (ISCA Journal), Vol.43, No.4, pp.311-329, September 2004.
- [5] Gordana Velikic. *The Use of Phase in Automatic Music Transcription* Doctoral Degree Proposal. University of Rochester. 2006.
- [6] Geoff Bartlett. *Genie: A First GA*, Practical Handbook of Genetic Algorithms: Applications. Volume I. Lance Chambers, Ed. CRC Press, 1995.
- [7] Conor Ryan. *Niche And Species Formation in Genetic Algorithms*. Practical Handbook of Genetic Algorithms: Applications. Volume I. Lance Chambers, Ed. CRC Press, 1995. 57-74