IMPROVING SONIFICATION TECHNIQUES USING GENETIC ALGORITHMS

BY

SEBASTIAN CAVE

A Thesis Submitted to the Faculty of the

DEPARTMENT OF COMPUTER SCIENCE AT WAKE FOREST UNIVERSITY

in Partial Fulfillment of the Requirements for

Graduation With Honors in

Computer Science

4/26/2023

Winston-Salem, North Carolina

Approved By:

Natalia Khuri, Ph.D., Advisor

Daniel Canas, Ph.D., Chair

William Turkett, Ph.D., Department Chair

Table of Contents

Abstract

Sonification is the representation of data using non-speech audio. Sonification techniques are not used very frequently, in part because they do not adequately represent the patterns within the data. Prior attempts to improve the melodic structure resulted in reduced accuracy of the sonification. In this project, a genetic algorithm with an expert-designed fitness function, is used to generate melodies for the sonification of tagged data. Experimental results demonstrate that genetic algorithm performs better than the naive method. Future work will test the effectiveness of this sonification method against visualizations techniques.

Chapter 1: Introduction

In a data-driven society, it is important to develop new ways for interacting with data. Better understanding of data will shape our civilization in the coming years, and as our knowledge of how humans perceive and understand the information grows, techniques used to aid this understanding are also changing. In John Heil's essay on Perception and Cognition, he reasons that bridging the gap between these two must occur through the invocation of the senses [4]. The most common method for data representation is visualization, which represents data by diagrams, for example. Visualizations are beautiful and effective, however other methods of representing data are largely underutilized.

Data sonification [2], for example, uses non-speech audio to convey information. Therefore, it holds great potential to increasing data understanding, in particular for visually-impaired users. Sound and music have been shown to improve pattern recognition and memory [1, 11]. Current methods of data sonification fall into one of two categories: waveform sonification or parametric mapping. Waveform sonification aims to use data to modify the shapes of sound waves [8]. Parametric mapping, on the other hand, maps data to different aspects of music, such as rhythm, tempo, or pitch [13]. I decided to focus my research on parametric mapping as I believe it to hold the most potential for increasing data understanding, in particular of labeled or tagged data.

Labeled or tagged data refer to the labels derived from the annotations of the raw data, helping to categorize and organize it. The best example of tagged data

is the part-of-speech (POS) tagging of natural texts. These tags are created by classifying each word within a body of text into categories based on their part of speech. Libraries, such as Natural Language Toolkit (NLTK), for example, allow users to generate such tagged data for any text [12].

Most common way of visualizing POS-tagged data is to present the tags as a list of tuples (word, tag), which can be visually overwhelming and difficult to process quickly. These data are challenging to sonify because they are not ordinal. What this means is that there is no comparative operation for the individual data points. The lack of this operation presents a problem for parametric mapping. When choosing to map data points in an ordinal dataset to something like pitch, the decision is easy: the smaller numbers map to the the lower pitches and the larger numbers map to the higher ones. But this is not possible for tagged data because there is no such thing as "smaller" or "larger" tags. Therefore, there exist many different ways to generate the mapping for tagged data. All permutations may produce a viable sonification that represents the data with a one-to-one mapping between the labels and the pitches. Therefore, the main research objective of this project is to determine computationally, which sonification out of all these permutations is the best.

Chapter 2: Methods

2.1 Genetic Algorithms

In this work, a genetic algorithm (GA) has been implemented to find the best melody for a tagged dataset. GAs are used to solve optimization problems using a strategy that models the process of natural selection [6]. The algorithm begins by generating an initial population of random solutions, and then based on the "fitness" of each solution, the algorithm selects a subset of the population with high fitness for reproduction, thus creating a new population. This process repeats until convergence, stopping when the average fitness of the population does not change between iterations. In this project, the initialization step generates a population of random mappings from the given tags to an octave of notes within the C major scale. These mappings are scored according to the melodies that they generate when applied to the sequence of tags in the given data. The better melodies have higher scores which allows us to maximize the score over a predetermined number of generations (Figure 2.1).

The melody score or fitness is a function based on the principles of music theory. Specifically, the function uses patterns that are commonly found within western melodies and comprises eight terms to score a melody [9]. The score is defined as follows: let S be the score, N be the length of the sequence, p_i be the ith pitch in the sequence, and $R_n(p_i)$ be a rule applied to a pitch in the sequence. $S = R_0(p_0) + R_1(p_N) + \sum_{i=1}^{N}(R_2(p_i, p_{i-1}) + R_3(p_i, p_{i-1}) + R_4(p_i, p_{i-1}, p_{i-2}) - R_5(p_i, p_{i-1}) -$

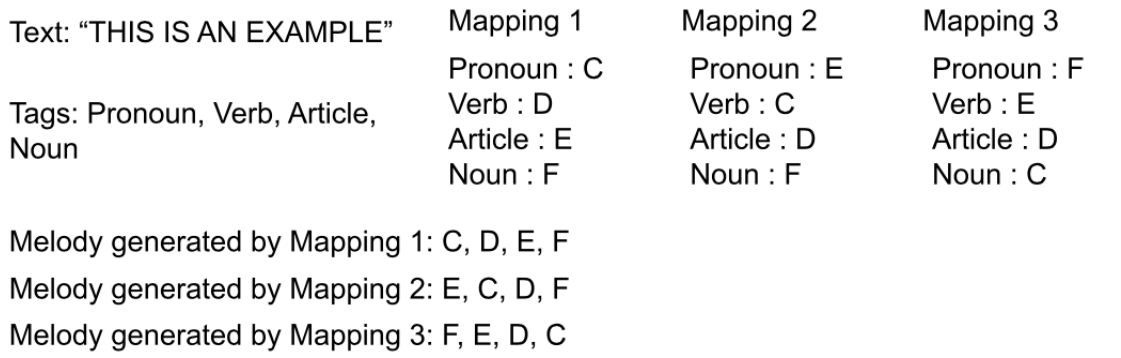

Figure 2.1: Example of three melodies generated for randomly-created mappings of a tagged sentence “This is an example.”

$R_6(p_i, p_{i-1}, p_{i-2}) - R_7(p_i, p_{i-1}, p_{i-2}))$. The individual terms of the scoring function are computed as follows.

- $R_0(p_0)$: Rule zero adds three to the score if the first note – p_0 – is the root of the key.

- $R_1(p_{N-1})$: Rule one adds three to the score if the last note – p_{N-1} – is anything in the root triad.

- $(R_2(p_i, p_{i-1})$: Rule two adds two to the score if the previous pitch is a single step away from the current pitch.

- $(R_3(p_i, p_{i-1})$: Rule three adds one to the score if the previous pitch is a third away from the current pitch.

- $R_4(p_i, p_{i-1}, p_{i-2})$: Rule four adds one one from the score if the distance between $i-2$ and $i-1$ is greater than or equal to a third, and the distance between $i-2$ and $i-1$ is less than the distance between $i-2$ and i.

- $R_5(p_i, p_{i-1})$: Rule five subtracts one from the score if the previous pitch is either a seventh away from the current pitch or greater than an octave away from the current pitch.

- $R_6(p_i, p_{i-1}, p_{i-2})$: Rule six subtracts one from the score if the distance between $i-2$ and $i-1$ is greater than or equal to a third, and the distance between $i-2$ and $i-1$ is greater than the distance between $i-2$ and i.

- $R_7(p_i, p_{i-1}, p_{i-2})$): Rule seven subtracts one from the score if the distance between $i-2$ and $i-1$ is greater than or equal to a third, and the distance between $i-1$ and i is also greater than or equal to a third.

In summary, each term either adds or subtracts a constant value from the total score based upon the sequence of pitches, allowing us to compute a fitness of each melody.

During evolution, GA uses genetic operators to improve the fitness of the population. These genetic operators include crossover and mutation operators.

The crossover operator takes two "parent" genomes and produces two "children" that share characteristics of both the parents. Similar to how evolution works in the natural world, this operator simulates the passing down of certain genetic characteristics through generations. Typical crossover operator is called a one-point crossover [7]. It chooses a random point within the chromosome of each parent and copies the genome of one parent from before that point and the other after the point. The point that determines how much of each parent genome to include in the children is called the crossover point.

One-point crossover will not work for our encoding as each solution is a permutation of a mapping between two domains. If the one-point crossover is used, it could generate mappings, where two notes are mapped to the same tag.

The alternative method that I devised retains the use of both parent genomes but safeguards against the case where a double mapping could occur.

For each tag, there are two options for the child to choose from: the pitch given by parent 1 and the pitch given by parent 2. A random choice is made between the two and we move to the next tag. The same decision is made here, however, if a pitch has already been used in the child genome, it cannot be reused and the other parent's pitch is selected. In the rare case that both parent pitches have already been used, the function selects a random unused pitch. This continues until all of the tags have been mapped within the child's genome.

The main goal of the mutation function is to ensure that the population does not become homogeneous and to prevent the algorithm from converging to local optima. The mutation function introduces random changes to the genetic material of the population. It is an important operator that helps to introduce genetic diversity into the population, which is essential for the algorithm's ability to explore a wide range of potential solutions. It is important to keep the rate of mutation within a GA relatively small. This is to avoid introducing too much randomness into the population and to ensure that the algorithm remains focused on promising solutions. The mutation rate for this project was set to one percent which means that one percent of the children that are being created will be put through the mutation function. The mutation function for our mapping performs a simple swap operation. The pitches that are mapped to two randomly selected tags are swapped to alter the

genome and the resulting melody.

To safeguard against convergence to a local optima, I implemented genetic reshuffling [5]. Once the fitness of a population has converged, take half the population, use the mutation function to change their mappings, and continue for another few generations. If the optimal solution had already been reached then those genes will persist and the fitness will re-converge to the same value. If not, then this genetic shakeup will provide another chance for the population to find a different solution without losing the progress that you have made so far.

2.2 Implementation Details

When given a tagged dataset, GA returns a sequence of notes to match the labels. The next step in the pipeline is a function that matches each note to a chord. This is done by looping through the melody a single time and assigning chords based on principles of music theory. Each note is assigned to a function: Subdominant, Dominant, or Tonic. The chords are then selected based on the scale degree and the function based on the chords of the "BIG 18", a collection of chords used within classical and folk music (Figure 2.2). Adding chords to the melody not only provides context for the notes, theoretically making the patterns more recognizable, but it also makes the music sound more full, which makes it sound better.

	$\hat{1}$	$\hat{2}$	$\hat{3}$	$\hat{4}$	$\hat{5}$	$\hat{6}$	$\hat{7}$
T	I		I^6		$I^{6}_{4}*$	vi*	
S	ii^{4}_{2}	$ii^{(7)}$		$ii^{6}_{(5)}$, IV		IV^6, vi*	
D		V^{4}_{3}, vii^{o6}		V^{4}_{2}	$V^{(7)}$, $I^{6}_{4}*$		$V^{6}_{(5)}$

Figure 2.2: The big 18 chords sorted by (T)onic, (S)ubdominant, and (D)ominant functions

The project was implemented in the Python programming language and developed in a notebook on Google Colab. The score function was constructed using the Mingus library [10], which provides MIDI and music theory functionality. The Python collections module and the random module were used to manage data structures and generate random values, respectively. The project was executed on a virtual machine provided by Google Colab. These machines have CPUs with 2 virtual cores, and 13GB of RAM.

Chapter 3: Results and Discussion

GA was executed on examples that keep the resulting melody within a single octave. This way I can use the brute force method to verify weather or not I have reached the optimal solution. The specific tags that I worked with to calculate my results the following:

[Noun, Article, Verb, Preposition, Preposition, Pronoun, Noun, Conjunction, Article, Pronoun, Verb, Verb, Pronoun, Adjective, Noun, Verb, Preposition, Preposition, Pronoun, Noun, Conjunction, Pronoun, Verb, Verb, Pronoun, Adjective, Article].

The average number of generations it takes to converge on the optimal value decreases with each population time. These averages were constructed from five test runs on the data listed above at each population size (Figure3.1). Increasing the population size comes at the cost of increasing the run time with the runs with a population size of 100 taking roughly ten times longer than the runs with population size 10. Note that these measurements are taken at the moment that the GA finds the value that it eventually converges on. This does not mean that it always found the optimal solution this quickly. The optimal solution was found, in this case, at a rate of 84% across all of the population sizes. Genetic reshuffling was disabled for these examples in order to get an idea of how long it takes for the GA to converge for the first time.

In order to test some of the strategies such as genetic reshuffling, I lowered the population size to ten in an attempt to try to get the fitness to converge on a local

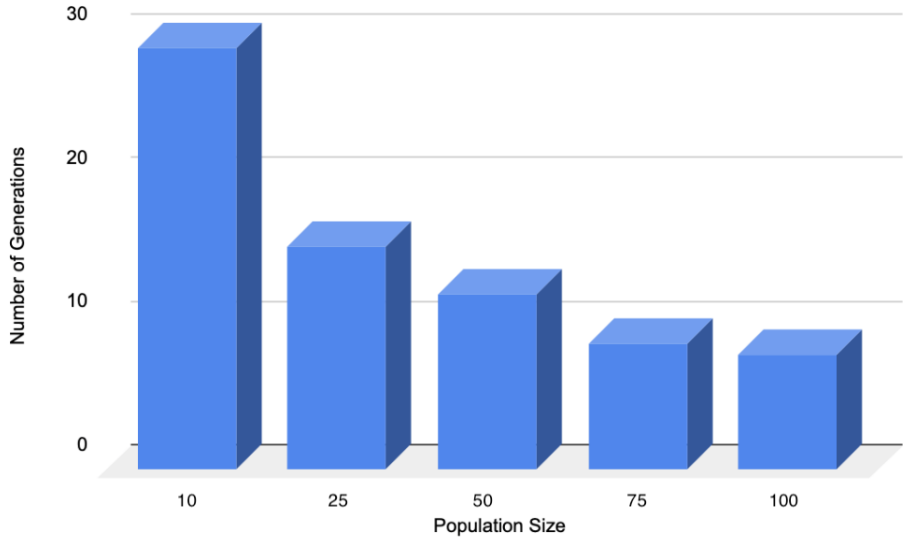

Figure 3.1: Average Convergence time for finding the best single-octave melodies.

optimum. I implemented it such that if there are three consecutive generations with the same average fitness value, a reshuffle occurs. We can see the average fitness value converging many different times before being shuffled into the correct optimal solution by the reshuffle (Figure 3.2).

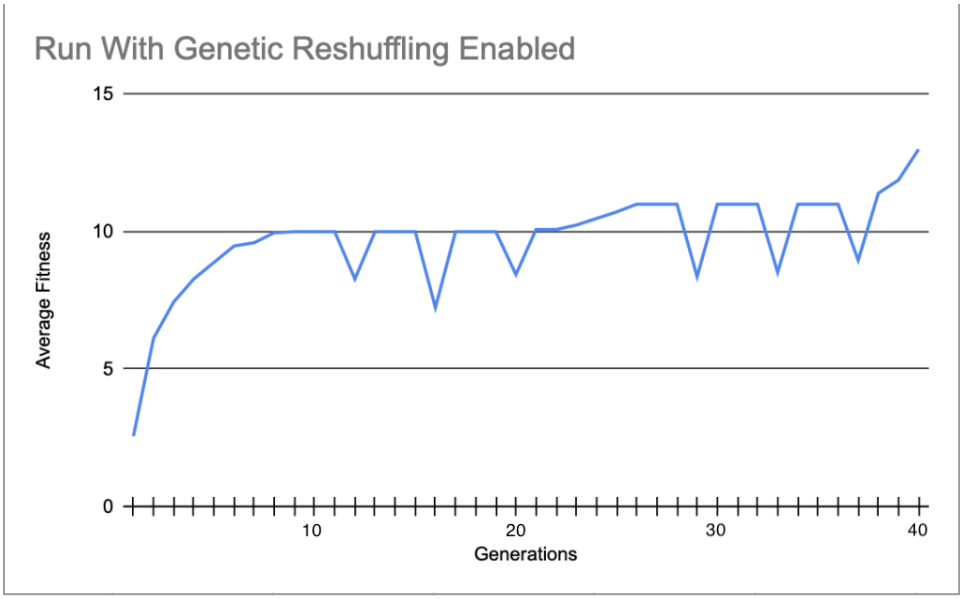

Figure 3.2: A test run where reshuffling is clearly helpful in eventually converging to the optimal solution of 13. Reshuffling can be seen in the graph with the sharp drops in average fitness. The fitness converges on 10, then 11, and then finally 13 after a good amount of reshuffling. The optimal solution is known here because I tested this small example using the brute force method as well.

Chapter 4: Discussion

An alternative to a simple GA is a multi-objective genetic algorithm (MOGA) [3], a strategy for solving problems that have multiple, conflicting objectives. In our case, the original scoring function could be divided into separate objectives, by splitting each term into its own objective, for example, and leveraging MOGA to improve the optimization process. The NSGA-II algorithm, for example, uses Pareto ranking to compare solutions by assessing how many objectives a solution is better at compared to its counterparts. In addition, NSGA-II generates new populations by appending parents' population to the population of children (Figure 4.1). This makes it so that the fittest members of the old population who most likely had higher scores than the lower scoring members of the new generation are able to directly pass their superior genes down to the next generation. This results in a higher average fitness each generation which can yield faster convergences on optimal solutions or local maximums.

There are some limitations of this approach to data sonification. It is not practical to use this strategy for datasets that contain a large number of tags. Due to the high numher of different notes used to construct melodies from this data, it becomes incredibly difficult to avoid things such as jumps of greater than an octave or sequences of many jumps in a row. For this reason, I recommend using this method on datasets that contain less than twenty-four tags which would be equivalent to the number of notes in two octaves.

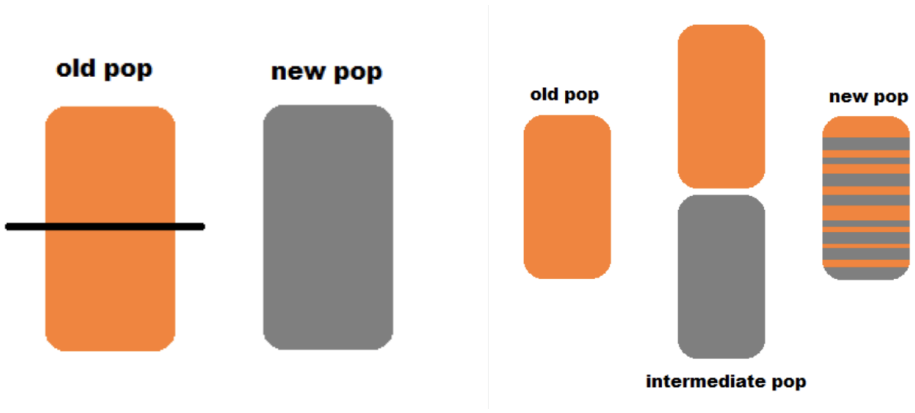

Figure 4.1: A visualization of the original strategy (left), and the NSGA-II strategy (right.)

Chapter 5: Conclusion

A simple Genetic Algorithm has been designed, implemented and tested for the purpose of sonifying tagged datasets.

Future work will explore the MIDI features of the Mingus library that was used to construct the scoring function. This will involve researching and experimenting with how to play and write notes onto a staff using the library.

Additionally, a user-study should be conducted to study if the patterns within tagged data are better represented by sonification compared to visualization. Participants would be given examples of each and then afterward asked to identify patterns in the examples. Such user experiments may reveal the gaps within existing implementation and venues for the creation of more specialized sonification.

Lastly, for text data, sentiment analysis could be added to the mappings. Music is good at evoking emotions, and sonification may be able to capture them. By changing certain qualities about the music, such as having a major or minor key, complex rhythms, or tempo changes, it would be possible to make the piece sound more sad, happy, angry, or even triumphant. This is a more nuanced addition that would be done by modifying the chord matching function. I think that it would only further enhance our ability to recognize patterns and understand what the sonified data represents.

Bibliography

[1] Ian Cross Beck Hanson. Why are song lyrics so easy to memorize? *The Naked Scientists*, 2013.

[2] Roger T. Dean. *The Oxford Handbook of Computer Music*. Oxford University Press, 2009.

[3] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[4] John Heil Fiona Macpherson. *The Senses, excerpt from Perception and Cognition*. Oxford University Press, 2011.

[5] Errin Fulp. Communication During Office Hours, Aug. 2023 2023.

[6] John H. Holland. Genetic algorithms. *Scientific American*, 267(1):66–73, 1992.

[7] Davie Lawrence. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

[8] Dale Parson. Mapping data visualization to timbral sonification and machine listening. *Computer Science and Information Technology Faculty Kutztown University*, 4, 2017.

[9] Alan Reese. Communication During Office Hours, Aug 2022 - Oct 2023 2023.

[10] Bart Spaans, 2015.

[11] Jessica Stoller-Conrad. Why do we remember countless song lyrics, but not our studies? *Figure One*, 2013.

[12] NLTK Team.

[13] Mark Ballora Teresa Carey. This musician transforms scientific data into elaborate melodies. *Freethink*, 2019.

Keywords— Genetic Algorithms, Data, Sonification, Data Understanding, Machine Learning, Tagged Data

Sebastian Cave
cavesr19@wfu.edu | 434.825.2084

EDUCATION

Wake Forest University Winston-Salem, NC
Anticipated Bachelor of Science in Computer Science // Minor in Mathematics *May 2023*
Major GPA: 3.833; Dean's List: Fall 2019, Spring and Fall 2020, Spring and Fall 2021, Spring and Fall 2022
Relevant Coursework: Natural Language Processing, Digital Media, Computer Systems, Multivariable Calculus, Discrete Mathematics, Linear Algebra, GPU Programming, Algorithms in Bioinformatics
Member of Upsilon Pi Epsilon: International Honor Society for the Computing and Information Disciplines

COMPUTER SKILLS/PLATFORMS

- Proficient with Python, C, C++, Java, and JavaScript
- Experience working in Unix-based environments and with Cloud Computing Services such as Azure
- Exposure to programming in CUDA, Assembly, and HTML

PROFESSIONAL EXPERIENCE

Elder Research Charlottesville, VA
Intern *Summer 2022*
- Collaborated with a team of other interns to develop creative solutions to complex open-ended problems
- Learned to efficiently communicate and work in a hybrid remote environment where only a portion of time was spent in the office
- Presented the results of the summer internship onsite, highlighting the importance and relevance of our work

Grade Potential Tutoring Winston-Salem, NC
Tutor *Spring 2022 - Present*
- Created a healthy learning environment to help students succeed in both Calculus AB and BC classes

Outdoor Pursuits, Wake Forest University Winston-Salem, NC
Trip Leader, Rock Wall Facilitator *Fall 2020 - Present*
- Led outdoor rock climbing, mountain biking, and hiking trips of 10-15 students
- Provided a fun and safe environment for students to learn new outdoor skills such as belay techniques, anchor building/cleaning, mountain biking, and fire-making

Fry's Spring Beach Club Charlottesville, VA
Senior Swim Coach *Summer 2020*
- Wrote and ran morning, afternoon, and evening practices that allowed swimmers to improve their technique
- Helped lead and organize 200+ swimmers ages 6-18 during meets and other events each week
- Contributed to a healthy competitive environment that led to the team being awarded the league's *Sportsmanship Award*

LEADERSHIP EXPERIENCE

St. Anne's Belfield Charlottesville, VA
Camp Counselor *Summer 2020*
- Designed and ran the *Game Design* camp for two weeks where students were able to use a variety of materials and methods to create original board games

High Performance Computing Team (Daemon Deacons), Wake Forest University Winston-Salem, NC
Team Member *Spring 2021 – Fall 2021*
- Worked in a team setting building and running software on the Wake Forest DEAC Cluster
- Competed in the international (SCC) Student Cluster Competition (held virtually due to COVID) building our applications remotely on Azure machines

Rock Climbing Club, Wake Forest University Winston-Salem, NC
Routesetter *Fall 2020 - Present*
- Collaborated with other routesetters to put up safe, interesting, and fun problems for all students to enjoy
- Developed critical and creative thinking skills through the creation and testing of complex routes.

Barrett Early Learning Center Charlottesville, VA
Intern / Assistant Teacher *Spring 2019*
- Learned to foster a healthy and safe learning environment for children in the Fifeville area ages 2-5

OTHER INTERESTS

- Rock Climbing, Nutrition/Athletic Training, Travel, Mixed-Media/Algorithmic Art