Explorations in Algorithmic Composition: Systems of Composition and Examination of Several Original Works

Jacob M. Peck
Candidate for Bachelor's of Science Degree in Computer Science

State University of New York, College at Oswego
College Honors Program

October, 2011

Abstract

This paper examines the concepts and history of algorithmic composition, as well as looks at several example programs written in an attempt to further engage with the material at hand. An overview of the important events in the development of algorithmic composition is given, alongside discussions of several topics important to the understanding of algorithmic composition. A collection of custom programs is presented which form the body of my hands-on experience with algorithmic composition, along with several example pieces from each. Areas for future study are identified, and several complications and weaknesses encountered in the research process are addressed.

Table of Contents

Contents

Advice to Future Honors Thesis Students

Writing this thesis was the furthest thing from easy that I could have imagined. However, and this may seem completely contradictory, this is also the smoothest example of writing I have ever put to paper thus far. I do believe that several things contributed to that, and it is my hope that if and when it comes time for you to write a thesis, that the following tips help you.

First and foremost, try your very hardest to complete all assignments in your section of HON 350 on time and before the semester is over. The organizational schedule they offer is a serious boon to thesis writers, and any student that is able to commit themselves to starting off on this path will be leaps and bounds ahead of their peers. This is a tall order, to be sure, as only three students (I think) in my section of HON 350 managed to complete all of the required work, out of a class of at least twenty. But hard work and determination pay off—believe me.

Second, make sure your topic thoroughly interests you. You’re going to be spending the next year and a half to two years researching every last nook and cranny of the topic, finding out what makes it tick, how to make it tick in a slightly different way, how to break it, why you might want to break it, how to build it back up again, etc. If you choose a topic that you’re simply not interested in, you *will* grow bored of it in the process, and either end up abandoning your thesis (only to start fresh, oh no!) or trudging through the pain and producing a sub-par work that neither you nor your advisors can be proud of. Pick a topic that is close to your heart, and better yet, don’t be afraid to break new ground. If you want to research the effects of global warming on declining seal populations and how this relates to the blubber industry, but are afraid because there is a lack of sources, don’t be afraid to draw your own conclusions! You are a scientist, a scholar, an academic, and your voice is important and should be heard! You will thank yourself, and your advisors will thank you for providing them something interesting to think about in the mean time.

Speaking of advisors, I firmly believe that a prudent choice of advisors is key to the production and motivation behind moving your thesis along. My two advisors, Professor Craig Graci and Dr. James Early, challenged me every time we met to push a little further, to seek a bit deeper, and to think, think, think! about what I was doing with regards to my research. And for this, I cannot be more grateful, as I am truly proud of what they helped me accomplish. They provided goals and sub-goals, and sometimes even sub-sub-goals, and in general were extremely supportive of me throughout the entire process. And as a side-effect, I like to think I've forged a great friendship with them, and I am extremely appreciative about that as well.

The best advice I can offer, however, is this: don't give up. The thesis may seem like it is a far way away, or an impossible monumental task, but believe me, it can be completed. Not only this, but you will find yourself better off for the experience. The process is grueling, to be sure, but it is even more rewarding in the end.

Acknowledgments

I would like to thank several people who contributed to this paper, either directly or indirectly.

Firstly, I would like to thank my mother for all she has done to support me in my life, academics and otherwise. Without her, none of this would have been possible.

I would like to thank my wonderful Honors thesis advisors, Professor Craig Graci and Dr. James P. Early, without whom, this endeavor would have surely fallen apart. The support I have received from these two in my research and my academic career has been absolutely amazing, and I cannot imagine any better advisors or mentors to have.

I would like to thank my friends and co-workers, who allowed me to bounce ideas off of them whenever I started rambling (sorry, guys!). In particular, I would like to thank Missy Hill and Ian Mumpton, both of whom contributed greatly to my creative process and for having an uncanny knack for finding bugs even if they're not technologically inclined. I would also like to thank Chrissy Brandon, Jess Tetro, Jenn Cartossa, Katie Lachut, Adam Sternberg, and Janel Sullivan for keeping my music alive. It truly means a lot to me, guys.

Lastly, I would like to dedicate this thesis in the honor of my father, Robert Charles Peck, and my grandmother, Linna Margaret Wilcox, both of whom passed away during the formative years of my life. My father, who passed away while I was writing this, and my grandmother both influenced me in ways the world may never know, and I have full intentions of making them proud. May you two forever play checkers. This one's for you!

Author’s Reflections

By motives opaque

My journey into algorithmic composition research was a rather natural progression due to various events in my life, combined with my natural interests. I have always been interested in music, as far as I can remember. This may be partly due to my childhood fascination with anything patterned and quantifiable, but I suspect it is a bit more primal and elementary than that. Music moves me, plain and simple.

Aside from being just a listener of music, I play several instruments as a hobby, but I wouldn’t say that I am particularly good with any of them, nor am I trained at all. I just pick them up and play, and record when the mood hits. However, I do believe that this ability to comprehend the intricacies of several different instruments further increases my understanding and appreciation of the musical space that each instrument occupies.

Though my favorite collection of genres of music lies squarely in the metal spectrum, I find almost any music worth listening to and understanding, and I have for quite some time. As a result, my personal music library (great times that we live in, that each individual may have a music library that belongs solely to them) contains several thousand works by a little over a thousand different artists in such diverse genres as folk, death metal, neo-classical, baroque, anti-folk, post-punk, soft rock, and even a few albums completely composed of ambient sound recordings. Music may be found anywhere. But I digress. Suffice it to say that music is my fuel.

The flame lights Prometheus

The first event that lead me down the path of becoming a computer scientist was when I around 7 or 8, and my father’s friend moved in with him for a brief period. This man was a computer fanatic. Whether or not he possessed the skills he claimed he did is another story altogether, but his discussions with me fascinated me. He was a genuine intellectual, or as much as you can be and still talk appreciatively to a 7-year-old. He never excluded me from conversation simply because I was too young to understand. No, he simply explained things in terms that I could understand and moved on to the next subject. Looking back on it, it was these conversations that helped to shape my future.

Moving beyond that, around age 15 or so I was a part of an engineering program hosted by Lockheed Martin, offering a free programming course in `C++`. I was hooked. Over the course of ten weeks, the class touched on the basics of programming, and led up to building a Yahtzee clone. Though it was simplistic work, it cemented my desire to continue programming.

While at SUNY Oswego, I opted to take a course titled “Cognitive Musicology” taught by Professor Graci. It was this course that eventually led to my research on algorithmic composition, due to the `MxM` interface and `Clay` programming language that Professor Graci uses to teach this course. It opened my eyes to the abilities of computers as musical machines, and allowed me to explore my ideas in a way that I was unable to previously.

Atlas takes a break

Well, that brings us to why I began this research, but it says little of what I learned and how I grew through the process.

In researching the topic and implementing various approaches to algorithmic composition, I have gained a substantial amount of mastery over certain concepts, in particular cellular automaton theory and Lindenmayer systems. These topics, though large and varied on their own, are much easier to digest when concrete examples generate themselves in the form of music. Whether or not the results were the best they could be did not matter. What I gained from the process is undeniably much more than the sum of the output generated by my programs. I gained `insight`, and that is something that truly has to be experienced to be understood.

Reading Nierhaus’ book (see [28] in the References section for more information) truly opened up my eyes to the incredible things people had done within the spaces where music and computing intersect. First let me say a small word about Nierhaus’ work: this is the single most comprehensive tome on any single subject that I have ever encountered, and yet it is still a casual reader, for those who are interested. As for the people detailed in the chapters of Nierhaus’ history of algorithmic composition, every story and every personality is interesting. From Hiller and Issacson to Cope to Xenakis to Miranda, it seems that any name attached to algorithmic composition has a unique approach to offer and a personal viewpoint that changes your perception of algorithmic composition as a whole. I firmly believe that it is the exploratory and experimental nature of these people that has influenced me to keep working on new ideas, even if they don't fill an immediate need.

The scribe uses Babbage's Difference Engine

The technology I've worked with in crafting this thesis was an incredible experiment in and of itself.

For the Java programs (`Wordsongs`, `HexMusicExperiments`, `Markov Machine`, `LCompose`, and `ALM`'s `JFugueDaemon`), working with JFugue and Gervill was a huge boon. Though Gervill isn't represented in this thesis, it did allow me to create some interesting renditions of the examples provided within the text by substituting in different soundbanks. JFugue, on the other hand, was central in creating the programs, as it allowed an incredibly simple, intuitive way of crafting musical building blocks with Java. Native support for working with the MIDI standard was nice as well. Learning a bit about MIDI files goes a long way, as evidenced by the chaotic and ugly scores of the `Wordsongs` examples, where I paid no attention to proper formatting, resulting in pieces that are unreliably translated. Compare those scores with the scores from, say, the `Markov Machine`, and you'll see a marked increase in the readability of these scores, as throughout the course of my experiments with JFugue, I became more aware of how to properly format a MIDI file.

`ALM` is a different beast altogether, being programmed in Common Lisp. `ALM` also has the distinction of being the largest project I've written, as well as the first modularized project I have undertaken. The flexibility and fluid nature of Lisp allowed me to do things I would have never been able to do with the rigid structure of strongly-typed languages like Java, while at the same time allowing me to express my thoughts in a concise, concrete manner. Lisp is by far my favorite language at the moment, and I oftentimes find myself using it to perform some simple calculation, instead of reaching for my calculator.

With this document, I have become intimately acquainted with LaTeX, a typesetting language that I highly recommend anytime someone asks me how to make a document look professional. LaTeX has provided me with nothing but a pleasure using it, and it takes care of all the nitty-gritty details for you, allowing you to get right into writing your document. Other word processing packages oftentimes have an interface that isn't all that intuitive, and gets in the way of writing. I like my word processing like I like my code editing, thank you.

Typesetting the music was an interesting aspect of this process as well. I settled on sending the MIDI files generated by JFugue through LilyPond, a music typesetting package. LilyPond generated .png (Portable Net Graphics format) images of the scores, which I then imported into this document. Though a bit indirect, it works without a hitch, and provides fairly decent results. There are music typesetting packages for $\LaTeX$, but I could not manage to get them working in my setup. LilyPad worked out of the box, so there is no reason not to use it.

for(;;){learnAndGrow();}

This thesis has been a tremendously rewarding experience for me. If it is never read by anyone beyond myself and my advisors, I would not mind, because the writing and research put into this has shaped me into a better writer, and a better scholar. I feel honored to have worked on this, simply because I was able to pour myself into it and receive so much in return. But this thesis is far from the end of my academic and personal growth. I know that in the grand scheme of things, I will have to constantly learn and relearn things in my life, and I couldn't be more excited for it.

Thanks for reading, and I sincerely hope you enjoy what follows.

$\longrightarrow$ *Jake, September 2011*

Thesis Body

1 Introduction

1.1 What is algorithmic composition?

The term “algorithmic composition” encompasses many different ideas. Gerhard Nierhaus has defined algorithmic composition as “composing by means of formalizable methods,” a definition which is broad enough to cover all of what could be considered algorithmic composition, and yet narrow enough to exclude more traditional methods of composition [28]. Another, more modern way of defining algorithmic composition is as computer and math music. Though this definition provides a basic idea of what algorithmic composition consists of, it fails to capture the full extent of what algorithmic composition is, as neither computers nor mathematics are required for the practice of composing a piece algorithmically. What algorithmic composition truly consists of, then, is wrapped within the term itself—composing music with algorithmic processes. This still leaves much to be desired, however.

There are many camps of algorithmic compositional thought, each with their own approaches to the concept, and these may be divided into two major schools. The first major school of thought in the practice of algorithmic composition is that of style replication. The style replication approach is exactly what its name entails—replicating style through the use of algorithmic processes. The style replicationist approaches to algorithmic composition tend to use heavily mathematical methods to perform statistical analysis upon a corpus of original works, in order to generate (with or without human interaction) a new piece of music in the perceived style of the corpus. Several popular and successful projects have emerged out of this line of thought, including David Cope’s `EMI`, short for “Experiments in Musical Intelligence.” `EMI`, given the proper corpus, has been able to compose new pieces in the style of Bach, Joplin, and even Balinese gamelan music [10]. However, despite its success, the style replicationist approach to algorithmic composition poses its own limitations. One common criticism of style replication is that the systems in use to produce these new pieces are inflexible and allow for little human interaction [36]. Likewise, the very nature of style replication allows little room for new and creative inspiration to take place, due to the statistics-heavy approaches these systems generally take.

The other major school of algorithmic composition is the school of original works. This school focuses on using algorithmic processes to compose original pieces of music in a style of the composer's choosing. These approaches are typically less statistic-heavy and focus instead on original ideas held by the composer to create a unique piece of music. From these efforts, many different approaches are taken in creating an interesting piece of music, that may not necessarily be aesthetically pleasing to listeners of tonal music but provide insights into how music works on a fundamental level. Several composers have made careers out of ideas from this camp of algorithmic composition, including architect-turned-composer Iannis Xenakis [13, 11].

1.2 A brief history of algorithmic composition

Algorithmic composition has an interesting history, spanning centuries, cultures, and ideas. The first recorded example of algorithmic composition comes from around 1000 A.D., with Guido d'Arezzo's automated method for creating chorale pieces out of bodies of text [28, 13]. Essentially, d'Arezzo assigned vowels to pitch classes within a key, and simply translated the vowels within the text into a chorus line to accompany the text [28]. This system was designed primarily for the use of churches to create hymns easily [28].

From these humble beginnings, an entire area of study arose. In 1650, Athanasiuys Kircher's work "Musurgia Universalis" contained the outline for an algorithmic composition system titled "Arca Musarithmica" [28]. This system utilized a number of engraved wooden sticks called *syntagmas* [28]. Through manipulation of these *syntagmas*, composition of material in five different musical styles was possible [28]. Again in 1661, Kircher created a device for a child called the "mathematical organ" which among myriad other uses, was able to compose rudimentary music based on the principles of what would later become set theory and type theory [28].

With the eighteenth century came the rise of musical dice games. Starting with Johann Phillipp Kirnberger's "Der allezeit fertige Menuetten- und Polonaisencomponist" ("The ever-ready minuet and polonaise composer") in 1757, these games typically revolved around rolling a pair of six-sided dice to determine which bar to select from a table of pre-written measures [28]. By 1812, at least twenty different examples of these musical dice games were in existence, from many different composers, including C. P.

E. Bach, Maximilian Stadler, and even Haydn and Mozart, though the latter two cases are unconfirmed [28].

The development of the computer created a great tool for algorithmic composition, and an explosion of efforts occurred in the following years. In 1955 and 1956, Lejaren Hiller and Leonard Isaacson created the first fully computer-composed work of music on the ILLIAC computer that the University of Illinois owned [28]. Titled the "Illiac Suite," this work was divided into four distinct parts, each using a different method of composition, culminating with the fourth section which deals with Markov analysis [16]. In 1963, Hiller and Robert Baker created the first interactive environment for computer-assisted music composition, titled `Musicomp` [28]. In 1970, Richard F. Moore's `Groove` allowed a way for a computer to deal with time signatures and other musical time measurements [28]. In the 1980s came `Midi Lisp` (an extension of the `Lisp` programming language to deal with MIDI manipulation), `Patch Work` (a graphical interface lying over top of `Lisp` designed for easing composition efforts), `Csound` (a `C`-derived music programming language), and `Bol Processor` (a system for real-time composition and improvisation) [28, 13]. The 1990s had a fair amount of algorithmic composition efforts as well, including `Open Music` (a `Lisp`-like visual programming language), `Common Music` (an extension of `Common Lisp`), and `Symbolic Composer` (a music programming language) [28, 33]. More modern approaches include `PureData`, `SuperCollider`, `Nyquist`, and `Max/MSP`, all of which are music programming languages [28].

The above history of algorithmic composition is simply a sketch of the most salient events in the development of the area. It is very important to note that all of these systems built upon the ones that came before them, and draw from vastly differing fields such as mathematics, logic, physics, and even economics. These approaches represent a moving tide of ideas that bring algorithmic composition to the position it occupies today. In the following sections, modern approaches to algorithmic composition will be addressed, particularly with respects to four different models of algorithmic thought—cellular automata, Markov chains, Lindenmayer systems, and genetic algorithms—in order to provide a foundation for the work to be presented later on.

1.3 Cellular automata

1.3.1 What are cellular automata?

Cellular automata are a subset of a field known as artificial life, which is itself a part of artificial intelligence. The goal of studying artificial life is to attempt to find solutions to problems that arise out of the interactions of discrete agents within systems made of relatively simple rules. These agents are often driven towards survival and away from death, and the ways in which they interact provide insights to the microworld in which they live, and may also provide clues to interactions within the real world [12].

Cellular automata are a class of artificial life simulations which are defined by being composed of a world of discrete cells (the agents) which are aware of their neighbors. Each cell may take one of many states representing a stage of the cell's "life," so to speak, and this state along with the states of the cell's neighbors determines which state the cell will take in the next generation [28, 37]. Most cellular automata are discrete cellular automata, meaning that the entire world—each and every cell—enters the next generation at the same time. There are some variations on this basic rule however, such as cyclic cellular automata in which each cell is iterated in a predefined order, or each row or column is iterated simultaneously but separate from the other rows or columns [28]. Another iteration strategy is the random cyclical cellular automaton, which is a cyclic cellular automata where the order is randomized by some source of entropy in between each generation. A third variation is the continuous cellular automata, also called discrete automata, whereby every cell is *continuously* iterating, and the entire system never comes to a definable generation, making the state of the entire system at any given point in time difficult to determine [28].

In terms of world geometry, several possibilities are common. Most common, perhaps, is the two-dimensional grid of cells, in which each cell may be located in space by an `(x,y)` Cartesian pair [37]. This topography lends itself well to "stitching" the world into a torus shape in three-dimensional space by attaching the left and right sides of the grid, as well as the top and bottom. In doing this, the world is continuous in all directions, but finite [28].

The one-dimensional cellular automata is defined as a single line of cells, each having exactly two neighbors (commonly denoted "left" and "right" for simplicity). Such a world may also be "stitched" so that the "rightmost" cell is a neighbor to the "leftmost" cell, creating either a ring or a Möbius strip, depending on personal interpretation.

Hexagonal two-dimensional cellular automata are uncommon but practical, as will be demonstrated in the section on `HexMusicExperiments` later. These cellular automata are composed of cells placed such that each cell has exactly six neighbors, and the world may be "stitched" by connecting opposite edges, though for most purposes this is unnecessary given enough room within the system.

Moving beyond two dimensions, three- and four-dimensional cellular automata exist, but are typically an extension of the two-dimensional grid where the additional dimensions act as a history of the previous generations [28].

In terms of what defines a neighbor, there are several definitions provided by various researchers. The von Neumann neighborhood (after John von Neumann, computing pioneer among many other pursuits), when referring to two-dimensional grid cellular automata, is simply defined as the four neighbors in the cardinal directions. The Conway neighborhood (after John Horton Conway, mathematician) expands this definition to also include neighbors in the diagonals. For one-dimensional cellular automata, Stephen Wolfram, a mathematician specializing in cellular automata, has defined the Wolfram neighborhood as the two cells directly adjacent to a cell, with the specification that the "left" cell is distinguishable from the "right" cell, rather than being lumped together. For hexagonal cellular automata, the neighborhoods are spuriously defined, but one common arrangement is an adaptation of the Conway neighborhood to a hexagonal arrangement, resulting in the six hexagonal cells directly touching a cell [28].

1.3.2 Brief history of cellular automata

According to research by Stephen Wolfram, the history of cellular automata as a concrete idea begins around 1951 with John von Neumann [43]. Interested in the idea of self-replication, von Neumann began to search for a system that could replicate itself. He initially thought such a system would require three dimensions, but after conversing with Stanislaw Ulam (an American mathematician) later decided that self replication was quite possible within two dimensions [43]. Following this conversation, he eventually devised between 1952 and 1953 a two-dimensional cellular automaton with an astonishing twenty-nine states per cell that was designed to mimic the reproductive aspects of both digital logic circuits and biological and chemical processes. With this system in place, von Neumann attempted to outline the creation of a machine within this world that would be capable of constructing a copy of itself [43]. His outline called for 200,000 cells, and it appears as though he did not believe that anything less complex would ever be capable of self-replication [43].

Out of von Neumann's original research, the 1960s saw increased interest in the self-replication problem. More and more increasingly simple cellular automata were devised that were capable of self-replication, and more interestingly, universal computation [43]. These renewed efforts in studies allowed researchers to draw parallels between cellular automata and computing [43]. However, by the 1970s, cellular automata had entered into a phase of recreational study rather than serious research. In fact, it is from 1970 that perhaps the most famous cellular automata, John H. Conway's Game of Life, was brought into existence [43]. Defined as a two-state, two-dimensional cellular automata—in which a "live" cell is "live" in the next generation if it has two or three "live" neighbors in the current generation, a "dead" cell is "live" in the next generation if it has exactly three "live" neighbors in the current generation, and any other cell is "dead" in the next generation—Conway's Game of Life injected some life into cellular automata study, but research seemed to be mostly for recreational purposes [43].

From this point onward, until around 1981 when Stephen Wolfram began studying cellular automata, the field had relatively little scholarly interest [43]. With the publication of Wolfram's first cellular automata paper in 1983, cellular automata received a renewed interest, and since then have been studied by increasingly more researchers [43].

1.3.3 Brief history of cellular automata based algorithmic composition techniques

Cellular automata based algorithmic composition efforts were first attempted by Peter Beyls. In 1989, he established a set of extended rules for one-dimensional cellular automata that might be usable in music generation [28]. This system used data input by a user to generate multiple voices of a melody. In 1991, Beyls allowed the user to interact with a two-dimensional cellular automata in real time to affect the musical output of the program [28]. This program also included self-regulating rule systems which prevented stagnation and over-stability—if the system was too stable, it modified its rules on the fly to create a more dynamic system [28]. Further, in 2003, Beyls' program `CA Explorer` utilized one-dimensional cellular automata along with Lindenmayer systems, and refinement through the use of a genetic algorithm, to produce a musical output [28]. The section on `ALM` below describes a system that I designed which uses a similar approach in combining these three algorithmic ideas, that was developed without knowledge of Beyls' work.

Following Beyls, Dale Millen produced `Cellular Automata Music` in 1990 which used the data output by cellular automata of varying dimensions and mapped these points to pitch and duration values to generate music [28]. Millen later expanded this work, and in 2005 described a new version of the software which implemented an extended version of Wolfram's one-dimensional cellular automata rules [28].

In 1993, another cellular automata based system arose from Eduardo Reck Miranda, titled `CAMUS` (from *C*ellular *A*utomata *MUS*ic, not to be confused with Millen's project above) [28]. This system uses two different cellular automata running in parallel—Conway's Game of Life and the so-called "Demon Cyclic Space"—to generate triads [28]. In `CAMUS`, Conway's Game of Life provides the pitch and duration information of the root note, while the Demon Cyclic Space provides the triadic structure through the use of a set of rules devised by Miranda. In the section on `HexMusicExperiments` below, a simplified approach to this type of project is set forth utilizing three parallel cellular automata of varying dimensions and rules to generate "chords" of loosely coupled notes.

1.4 Markov chains

1.4.1 What are Markov chains?

Markov chains belong to the fields of statistical analysis and probability. A Markov chain (also known as a Markov model) takes the form of a matrix that demonstrates with which probability a system at the current time t with the current state will enter a given state at the next time interval, $t + 1$. The difference between Markov models and more cut and dried probability charts is that Markov models work stochastically, to generate probabilities for the next state given a number of previous states. For instance, Nierhaus demonstrates a simple Markov model for weather prediction, which based upon today's weather provides probabilities for the weather tomorrow [28].

Markov models are classified as nth-order Markov models, where n is the amount of previous states the probabilities depend upon. The previous example of the weather prediction model is a first order Markov model, as it depends solely upon the weather today—one previous state. Markov models of any arbitrary number of states are possible, though as a rule the higher the order of a Markov model, the more specific and inflexible its output becomes [28, 39].

A subclass of Markov models are known as hidden Markov models, where the output of the model is plainly observable, but the internal workings of the system itself are obscured. Such a system may occur in a situation where not all agents in the system are aware of the states of the others. To extend upon the weather prediction example above, the model may predict that there will be a thunderstorm tomorrow, even given sunny skies today, as there may be hidden information like barometric pressure informing the model that the user is unaware of. Hidden Markov models make for obscured, informed predictions that make sense in a fuzzy logic sort of way [28].

1.4.2 Brief history of Markov chains

The development of Markov chains seems a bit convoluted, but through tracking papers published from the eighteenth through twentieth centuries, a sort of mathematical lineage may be uncovered. In 1738, a Russian mathematician named Daniel Bernoulli wrote a paper on probability theory—the first written in the Russian language on the topic—planting the seeds for mathematicians in Russia to begin exploring the area of probability [3]. However, this field went unnoticed for a little over a century until 1846, when V. Y. Bunyakovsky wrote the first textbook in the Russian language on probability theory [3]. Also in that same year, the first Russian dissertation on probability theory was completed by Pafnuty L. Chebyshev at Moscow University [3]. Chebyshev went on to become Andrei Andreevich Markov's teacher and mentor [3].

Markov, from whom Markov chains get their name, built upon the works of several mathematicians before him, even if Markov himself was unaware of the fact. In 1900, the first edition of Markov's seminal textbook "Calculus of Probabilities" reached publication [3]. With this, he established himself in prime position to begin upon more serious theoretical work. By 1906, he had written (and in 1907 had published) his first paper on "simple chains" which consisted of only two states, 0 and 1 [3]. This paper had never used the term "Markov chain," which appeared first in a 1926 paper by S. N. Bernstein [3]. After this paper's publication, Markov's colleague and correspondent A. A. Chuprov pointed out in a letter to Markov several predecessors of the same theories which Markov had arrived at, including work by D. Bernoulli, Laplace, Ehrenfests, the concept of Brownian motion, Bunyakovsky's random walks, and Bachelier's study of the stock exchange [3]. Markov, apparently unaware of these previous works, began to study them whenever he could procure their articles, and as a result began to refine and redevelop his chaining theory [3].

In 1913, the third edition of "Calculus of Probabilities" was published, containing the first real-world application of Markov's chains [3]. Markov analyzed the probabilities of vowels following vowels and vowels following consonants in two staggeringly large works: A. S. Pushkin's poem "Eugeny Onegin" (consisting of 20,000 letters) and S. T. Aksakov's novel "The Childhood of Bagrov, the Grandson" (consisting of 100,000 letters) [3]. Being the early twentieth century, Markov would have had all of this calculation by hand, but that did not prevent him from providing this persuasive example [3]. Markov found that in Pushkin's poem, any given letter had a probability of 0.432 of being a vowel, and through the use of his chains, he found that there was a probability of 0.128 that a vowel would follow another vowel, and a probability of 0.663 that a vowel would follow a consonant [3]. In Aksakov's work, he found a probability of 0.449 that any given letter could be a vowel, and through the use of his chains again, found a probability of 0.552 that a vowel would follow another vowel, and a probability of 0.365 that a vowel would follow a consonant [3].

1.4.3 Brief history of Markov chain based algorithmic composition techniques

Markov models in algorithmic composition frequently fall into the style-replicationist school. Harry F. Olson was the pioneer of using Markov chains to compose, in 1950. He used Markov analysis to generate first- and second-order models of melodies by Stephen Foster. The resulting models show a much increased ability to compose coherent and harmonious pieces when comparing the second-order model with its first-order counterpart [28].

The 1950s held several other Markov-related algorithmic composition efforts. In

1955, Lejaren Hiller and Leonard Issacson began work upon their “ILLIAC Suite,” widely regarded as the first completely computer-composed work of music [28]. In the fourth part of the suite, Markov models of varying orders are utilized to generate musical structure [28]. Iannis Xenakis, an algorithmic composer, used Markov models beginning in 1958. This led to the work “Analogique A,” in which Markov models are used to arrange the segments of different sonic density [28]. Xenakis also used Markov models in the pieces “Analogique B” (1958) and “Syrmos” (1959) [28].

Also of note is the 1993 effort of F. P. Brooks, A. L. Hopkins, P. G. Neumann, and W. V. Wright. These four used a body of thirty-seven chorale melodies (all in common time which contain no notes shorter than an eighth) to produce a number of Markov models of varying degrees (between first- and eighth-order), and subsequently utilized these models to generate compositions [28]. One important result of this work is concrete evidence for the problems of Markov models in algorithmic composition: using too low of an order tends towards incoherence and randomness, while too high of an order runs the risk of simply rewriting the piece without offering any insights or creative differences between the source and the finished product [28].

Hidden Markov chains have had their fair use in algorithmic composition as well. In 2000, Martin Hirzel and Daniela Soukup used a hidden Markov model to generate jazz improvisations from patterns entered into their program by the user. These patterns are modified based upon previous training by a specific piece (“Forest Flower (Sunrise)” by Charles Lloyd) through an algorithm whose implementation is hidden from the user [28]. Mary Farbood and Bernd Schoner in 2001 utilized hidden Markov chains to generate counterpoint in relation to a melody entered by the user. The program processes this melody via a hidden algorithm, and produces an output consisting of a counterpoint line [28].

1.5 Lindenmayer systems

1.5.1 What are Lindenmayer systems?

A Lindenmayer system is a formal generative grammar with certain integral constraints upon their implementation [28]. Defining an L-system involves defining a three-tuple of values: the alphabet (a list of symbols which acts as both the input and the output lexicon for the grammar), the transformations (a set of rewrite rules defined in the following manner: `<input symbol>` $\rightarrow$ `<output symbol 1> ... <output symbol n>`, where each symbol on the right hand of a rule is defined in its own unique rule, such that all symbols in the alphabet are defined in terms of only one rewrite rule), and an axiom (a symbol or string of symbols from the alphabet used as the initial input to the system) [28, 38]. In contrast to Chomsky's context-free grammars, each iteration of an L-system acts on the entire input string in parallel, as opposed to the traditional piece-wise sequential approach [28, 9].

Due to the definition of the three components of an L-system as described above, L-systems contain no terminal symbols, although terminal symbols may be emulated by defining a symbol in relation to itself (*i.e.* `R` $\rightarrow$ `R`). Because of this fact, L-systems tend to generate self-similar patterns which are of great use in music composition [28, 38]. L-systems are also completely deterministic, due to the strict one-to-one mapping between symbol and rewrite rule. Given the same rules, axiom, and number of generations, the exact same output will be generated every time. However, despite being completely deterministic and self-similar, it is rather hard to predict what one given L-system will generate some number n generations in the future simply by looking at the rules, lending a sense of non-transparency to them.

1.5.2 Brief history of Lindenmayer systems

L-systems, as mentioned above, belong to the class of context-free grammars as described by Chomsky. Chomsky, a linguist by training and article-producing powerhouse by hobby, had a very healthy interest in what makes a language definable, particularly with regards to syntax. In the early 1950s, he began his research into the structure of language and syntax, eventually coming upon the idea of grammars. These grammars, or discrete systems of rules to generate parsable sentences, come in many classes, as Chomsky discussed. By 1965, he had published a paper containing a discussion of the class of context-free grammars, in which a string of symbols is parsed and transformed according to a set of rules without regard to context [9].

L-systems arose out of a desire to model the growth of certain biological life. Astrid Lindenmayer, a Hungarian botanist and biologist, was searching for a way to model and analyze growth of various plant life, including algae and ferns [32]. In his search, he hit upon the realization that a modified version of Chomsky's context-free grammars could fit the bill easily, as these systems produce the much needed self-similarity and regularity that nature tends to impose. By using a turtle-graphics approach to visualizing the output of these systems, Lindenmayer had everything he needed to begin analyzing the growth of plant life [32].

1.5.3 Brief history of Lindenmayer system based algorithmic composition techniques

L-systems in algorithmic composition are a varied topic. Roger Luke DuBois, in his dissertation in 1996, described several different mapping strategies for using L-systems to generate music. One of his mappings is a moving mapping, whereby the output of the system is transformed according to a table which changes after every symbol is transformed [28]. DuBois also experimented with using a string of output to generate a chord, by defining two symbols (one for note-on and one for gap) and interpreting each successive symbol as a semitone raise on the chromatic scale [28]. In a third attempt, DuBois creates an L-system equivalent of the rule 30 Sierpinski triangle cellular automaton, and mapping the output of this L-system directly to staff notation to generate music in the shape of a triangle, of sorts [28].

Hanspeter Kyburz, in 1993, wrote a piece for saxophone and ensemble titled “Cells” [35, 13]. Kyburz uses the first thirteen generations of an L-system he devised to select pre-written segments of music, and simply concatenated them, one after the other, to create “Cells” [35, 13].

Stelios Manousakis wrote his Master’s Thesis on “Musical L-Systems” in 2006 [24]. In this work, he presents a framework for utilizing L-systems to generate musical structure in many different ways, including revisiting work by DuBois [24].

1.6 Genetic algorithms

1.6.1 What are genetic algorithms?

A genetic algorithm is an appropriation of the ideas of Darwinian evolution to the domain of problem solving [27]. A genetic algorithm works by simulating evolution on a population of individual solutions to a problem to arrive at a selection of most-fit solutions [28].

There is no hard and fast definition as to what an algorithm must do to be considered a genetic algorithm, but several ideas are common amongst varying implementations. First among these is the concept of the individual. A population of these individuals is created, often randomly, as an initial population for the model of evolution to work upon. Each of the individuals represents a potential solution to whatever problem the genetic algorithm was devised to solve. Typically the individuals hold a payload describing their solution, which in many cases is simply a string of characters of some sort [27].

The fitness function is a very important part of any genetic algorithm, as it will determine whether or not the system works to appropriate a solution. The fitness function operates on individuals, assigning them a score (many times on a scale between 0 and 1, but not always) depending on how well their payload solves the problem. An oversensitive fitness function can completely throw off the results of a genetic algorithm, as can one that is not sensitive enough. A genetic algorithm that doesn't produce decent solutions to the problem it is trying to solve can often be traced back to a poor fitness function [27].

A number of genetic operators are at work in most genetic algorithms as well, including the basic copy operator (an individual is simply copied over to the next generation intact), the crossover operator (two individuals are chosen to "mate" to create a new solution for the problem, often implemented by replacing one part of the payload of the first individual with a part of the payload of the second individual), and the mutation operator (an individual's payload is changed slightly as it is placed into the next generation) [27, 28]. The individuals selected as candidates for these operators are often chosen through a process known as preferred selection, which involves taking a certain number of random samples from the population and choosing the one with the best fitness. This process helps ensure that the average fitness of the population increases with each generation, rather than regressing.

1.6.2 Brief history of genetic algorithms

The forerunners to modern genetic algorithms first appeared in the late 1950s. Evolutionary biologists, attempting to model biological evolution computationally, created small simulations of evolutionary processes, but did not make the leap from simulation to problem solving [25].

In 1962, researchers G. E. P. Box, G. J. Friedman, W. W. Bledsoe and H. J. Bremermann, working independently of one another, had arrived at similar ideas for evolution-based problem solving [25]. In 1965, Ingo Rechenberg of the Technical University of Berlin demonstrated an early genetic algorithm. Though lacking several features of modern genetic algorithms—the simulation had no concept of a population, but instead consisted of a single parent agent that underwent mutation into two offspring that were rated for fitness, with the best being selected to act as the parent agent in the next generation—this simulation showed the potential of evolutionary search [25]. In 1966, researchers L. J. Fogel, A. J. Owens and M. J. Walsh improved upon Rechenberg's process by replacing the agents with finite state machines, creating perhaps the first instance of genetic programming [25].

Modern genetic algorithms, however, owe much to the work of John Holland. It was Holland who first suggested the crossover and other recombination operators, now common features of a genetically-flavored program [25]. With his 1975 textbook "Adaptation in Natural and Artificial Systems," Holland established the first fully-developed framework for a genetic algorithm implementation in an artificial system, creating the model that is still followed, more or less, today [25].

1.6.3 Brief history of genetic algorithm based algorithmic composition techniques

Genetic algorithms in algorithmic composition have a short but interesting history, largely lying in the school of style replication. Andrew Horner and David Goldberg described generating melodic material through the use of genetic algorithms in 1991, marking one of the first attempts at using genetic algorithms for music composition [28]. Bruce L. Jacob's program `Variations` uses genetic algorithms in three different modules ("composer," "arranger," and "ear") to attempt to replicate the composition process that a human composer would carry out [28]. Ryan Mc Intyre's "Bach in a Box" system attempts to harmonize with existing melodies using genetic algorithms [28]. John A. Biles utilized genetic algorithms to generate jazz solos [7].

Rhythm generation is not left out, however, with a few notable projects. Damon Horowitz developed a system where rhythmic segments are generated and rated by a user, in a sort of supervised genetic algorithm [28]. The work of Alejandro Pazos,

A. Santos del Riego, Julian Dorado, and J. J. Romero-Cardalda uses two interacting populations in a genetic algorithm model which is rated for fitness with an artificial neural network and input from a user [28].

2 Background

Throughout this paper, a number of concepts are referenced. What follows is a brief introduction to the salient topics of interest to you, the reader.

2.1 Musical concepts referenced throughout

2.1.1 Rhythm

Rhythm is the relation of durations between a series of notes [22]. The duration of a note, for example a quarter note, defines for how long the note should be continue to play after the initial attack. Through manipulating the durational relations between notes, a composer is able to create differing rhythms in a piece which hold the listeners attention, or in some cases, challenge the listener to make sense of a particular part of a piece.

Rhythm allows for a nice way of grouping a piece into a collection of similar elements, and helps the human mind to distinguish different forces acting throughout the course of a particular piece [22].

2.1.2 Meter

Meter is the term that most people mean when they think of or discuss rhythm, and the distinction between the two is a bit fuzzy at times. Meter arises out of a combination of rhythm and loudness [22]. Through the interplay of these two facets of music, a sense of time and direction are able to be established, creating what many recognize as a time signature [22]. From varying rhythmic intervals between individual notes but arranged in a repeating manner, with cues from loudness, a time signature may be derived for a section of a piece, allowing the brain to comprehend the stream of information more clearly and concretely. Take, for example, a waltz. The waltz meter is essentially grouping notes through the use of rhythm into threes (“one-two-three, one-two-three”) [22].

This temporal grouping allows the mind to interpret the various sections of a piece in coherent chunks, leading to a much greater appreciation of the music, as compared to disparate noise events [22].

2.2 Other concepts of interest

2.2.1 Aesthetic

The philosophy of aesthetics revolves around the question “what counts as art?” [14]. The study of aesthetics, then, seems to be the study of what qualifies as an artistic experience. But this is ethereal and vague. An ‘artistic experience’ must mean something in order for aesthetics to mean anything at all. So, it would do well to establish what ‘art’ seems to be at its core.

While the list of attributes of art vary wildly from person to person, a few basic principles may be reached, that while not universally acceptable are at least understandable nearly universally. One attribute on this list might very well be beauty—be it directly observable or subtly conveyable in content—for a piece of art needs an element of beauty in some form to draw the attention of its audience. But beauty is not enough, as there are pieces of art which few people would consider beautiful but are enjoyed by many still. Perhaps another attribute of art is the emotional engagement with the work. Without an emotional attachment, the work falls into the realm of “pretty pictures” and out of the world of “art.”

Aesthetics must comprise the study of these two aspects of a piece of art—beauty and emotionality—and in some light be a judge for what is considered art. These same criteria are also easily applicable to music.

2.2.2 Emergence

Emergence is the idea that a small set of simple rules will result in complex behavior [12]. The interactions between cells in cellular automata, the simple evolutionary idea behind genetic algorithms, and even the output of the simplest L-systems may all be considered emergent behavior. Emergence in this facet is helpful to attempt to model the real world, in order to gain insights that are otherwise inaccessible. One of the first scientific uses of the idea of emergence was Craig Reynold's `Boids` system, used to model bird flocking patterns [12]. In this model, each individual "boid" had two simple rules to follow—fly towards the center of gravity of the entire flock, and avoid colliding with other boids and obstacles—which when applied to the entire flock, led to very similar flight patterns to those of real-world birds [12]. These simple rules also allowed the flock to avoid obstacles when they were placed in the flight path of the boids. With no additional rules added, the boids split into two smaller flocks to fly around the obstacles and rejoined on the other side [12].

The emergent behavior of the boids may be a far cry when speaking about music, but music has emergent properties as well. Take, for example, the idea of a pedal point—a single note repeated at a consistent interval, fairly regularly—and its affect on a piece of music. For a familiar example, take the song "Blackbird" as performed by The Beatles. This piece uses a pedal point of a G note, played almost consistently every other note for most of the song. This simple repeated note, however, ties the music together, resolving many of the uneasy accidentals that McCartney wove into the piece, and at the same time provided a very strong basis for the meter of the song, acting as a metronome. The simple rule of repeating a G note every other note provides coherence and meter to a piece that might otherwise be a sea of less substantial noise.

2.2.3 Discussion on "The math behind the music"

There is a lot to be said about math and music. Many mathematical models are able to capture the mechanics of sound and the interplay of harmonics and dissonance [15, 6, 42]. What follows is a brief explanation of the very basics of acoustics and the more mathematically-inclined composition techniques of harmony and counterpoint.

2.2.3.1 Acoustic theory Music is expressed through the medium of sound. Sound takes the form of varying waves of air pressure [20]. These waves may be expressed by the formula $f = \frac{c}{\lambda}$, where f is the wave's frequency, c is the wave's velocity, and λ is the wave's wavelength [20]. By knowing any two of these three pieces of information, the third may be deduced. As the frequency increases (due either to the wavelength λ

decreasing, or the velocity c increasing), the audible pitch also raises, demonstrating that pitch is directly related to frequency. A frequency of 440Hz is commonly accepted as "concert *A*," and acts as a very good reference point to tune instruments.

Octaves of pitch classes occur at multiples of each other. For example, given the $A440$ example above, waves with frequencies of 110Hz, 220Hz, and 880Hz would all be considered A notes [15, 42]. However, the sound that one hears from a particular instrument—the timbre (pronounced *tamm-burr*)—is an emergent sound due to what is known as the harmonic series [15]. When a string on a guitar is struck, for example, not only is the intended note given off, but also a collection of harmonic waves in varying proportions of the frequency of the original note. It is the collection of these waves and their relative concentrations that lead to the differing sounds of different instruments [22].

2.2.3.2 Harmony and counterpoint

Harmony and counterpoint are two different, yet related studies in music composition. Harmony is the study of relations between intervals and chord progressions [4]. These relations, if properly applied, produce pleasing arrangements to the ear. Counterpoint is the study of how two or more independent melodic lines interact and harmonize with each other [5]. Counterpoint is clearly evidenced in Bach's "Crab Canon," a piece which consists of one melodic line which is then reversed and played concurrently with the original line. These voices react with each other in an interesting way that creates a layered effect to the music, and feels as though both lines are driving towards their own motives. In many cases, both of these topics rely heavily upon mathematics, even if it is not readily visible to the composer [15, 6].

2.2.4 Modernist music

2.2.4.1 Atonality and serial music

Atonal music is a major departure from more traditional, tonal music. In tonal music, pitches belong to scales, which resolve in various well-defined ways to other pitches. Not so, in atonal music, where the idea of a scale is mostly thrown away in favor of music that employs pitch classes without regard to antecedents [11]. Atonal music is often disconcerting to the practiced listener of more traditional western tonal music, with early pieces by composer Arnold Schoenberg even leading to riots and injuries upon their first performances [11].

Serial music, a creation of Schoenberg's, is a very strong example of atonal techniques [11]. Serial music involves using what is known as a twelve-tone row, or an ordering of the twelve pitch classes in the chromatic scale, and applying operations upon the row to produce variants [11]. From these rows, several methods are used to piece together a work, including repetition, transposition, and concurrency [11]. The net effect of these operations is a piece of music in which there is no tonal center as all pitch classes hopefully occur with the same frequency, and also few notes repeated in close proximity to each other, preventing even a momentary key.

2.2.4.2 Minimalist music Minimal music is much more tonal than serial music, and often is easier for a practiced western listener to appreciate. Minimalism is oftentimes heavily influenced by eastern ideas, such as the structure of the Indian *raga* form [11]. One common theme among minimalist composers, particularly Philip Glass and Steve Reich, and to a lesser extent John Cage, is a reliance upon repetition [11]. Often, a minimalist piece consists of a small snippet of written music and a body of performance instructions. Philip Glass' *Two Pages*, for example, consists of one and a half pages of written music in an incredibly fast tempo, but due to the repetition requirements, a performance of *Two Pages* often runs into the twenty-minute mark. Minimalist composers experiment with differing instrumentation, as well as with traditional musical structure, with many works ending where they began and using instruments such as telephones, tape recorders, and humans clapping. John Cage's most famous work, *4'33"*, is an extreme example of minimalism, minimizing everything about the piece, including the sound itself. Consisting of four minutes and thirty-three seconds of absolute silence, Cage created a piece that questioned the very essence of musical aesthetics.

3 Creative works

I undertook several different approaches in exploring algorithmic composition, each leading to their own discoveries and challenges. Discussed below is a selection of five of thse approaches, along with examples and analysis of pieces composed through these approaches.

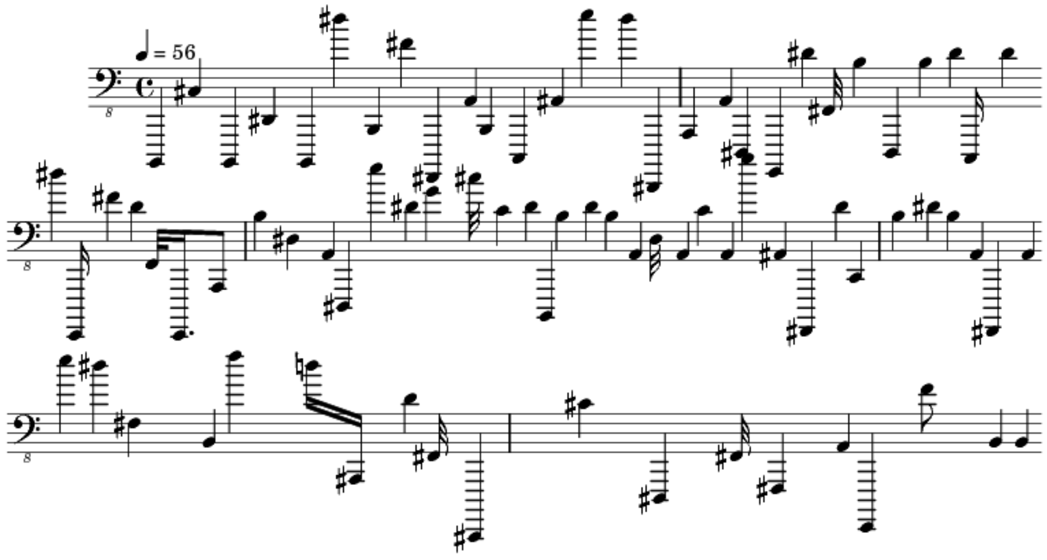
3.1 Wordsongs

3.1.1 Description of Wordsongs

`Wordsongs` is a quick little program designed as an introduction to computer-based algorithmic composition. Written in Java and utilizing the JFugue library [21], `Wordsongs` produces music by reading in a text file, performing some calculations based upon the contents of that file to map words to notes, and producing a MIDI file of the output. This is accomplished by reading the input file one “word” (whitespace-delineated token) at a time. The word is hashed, and the absolute value of that hash, modulo a constant (defined to be 40, but arbitrarily decided upon by the designer), plus an offset constant (defined to be 30, arbitrarily) results in a midi note value, ranging from 30 ($F\sharp 2$) to 69 ($A5$). The duration of the note is determined by the length of the word, modulo 32, plus 1, all over 32, to produce a fractional duration value between $1/32$ beat and a whole beat. This note and duration are then concatenated to the end of the current piece, and the program continues in this manner until the input file has no more tokens, at which point the piece is audibly played and saved to a MIDI file.

3.1.2 Pieces composed using Wordsongs

3.1.2.1 Wordsongs.java.midi

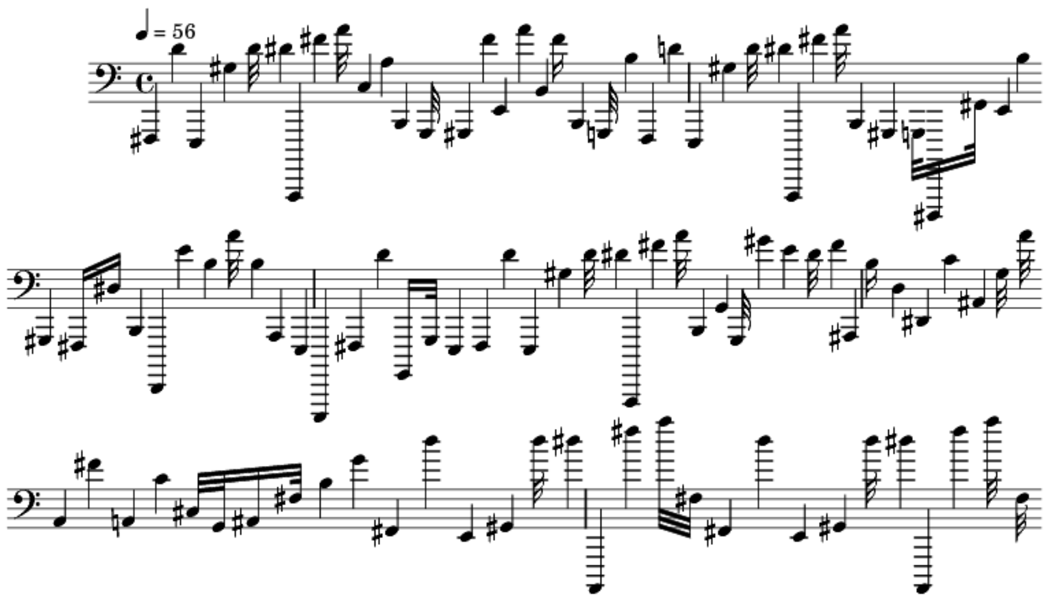


This piece was generated by feeding the source code of `Wordsongs` to itself. As can be determined from the score above, the output is relatively chaotic, leading to a disconcerting feeling to the piece itself. This particular piece seems rather atonal, lacking in any sort of resolution to a tonal center, however it still maintains a minor character of arpeggiation at certain points as well as a loose feeling of coherence.

The chaotic distribution of pitches and durations may be ascribed to the unique, uneven structure and distribution of characters and whitespace in Java code. Due to the way `Wordsongs` processes these tokens, almost every token in a source file such as this will result in a unique note. The fact that some of the notes play well with each other is simply a result of the constraints placed upon the system along with a bit of luck to create a small case of emergence.

As a musical object, this particular piece may be considered interesting simply because it contains qualities of both tonal and atonal music, while at the same time neglecting to adhere to the constraints of either style. As an experimental result, this piece demonstrates that `Wordsongs`, while chaotic and extremely simple, is a worthwhile diversion while considering algorithmic composition as a whole.

3.1.2.2 Twinkle.txt.midi

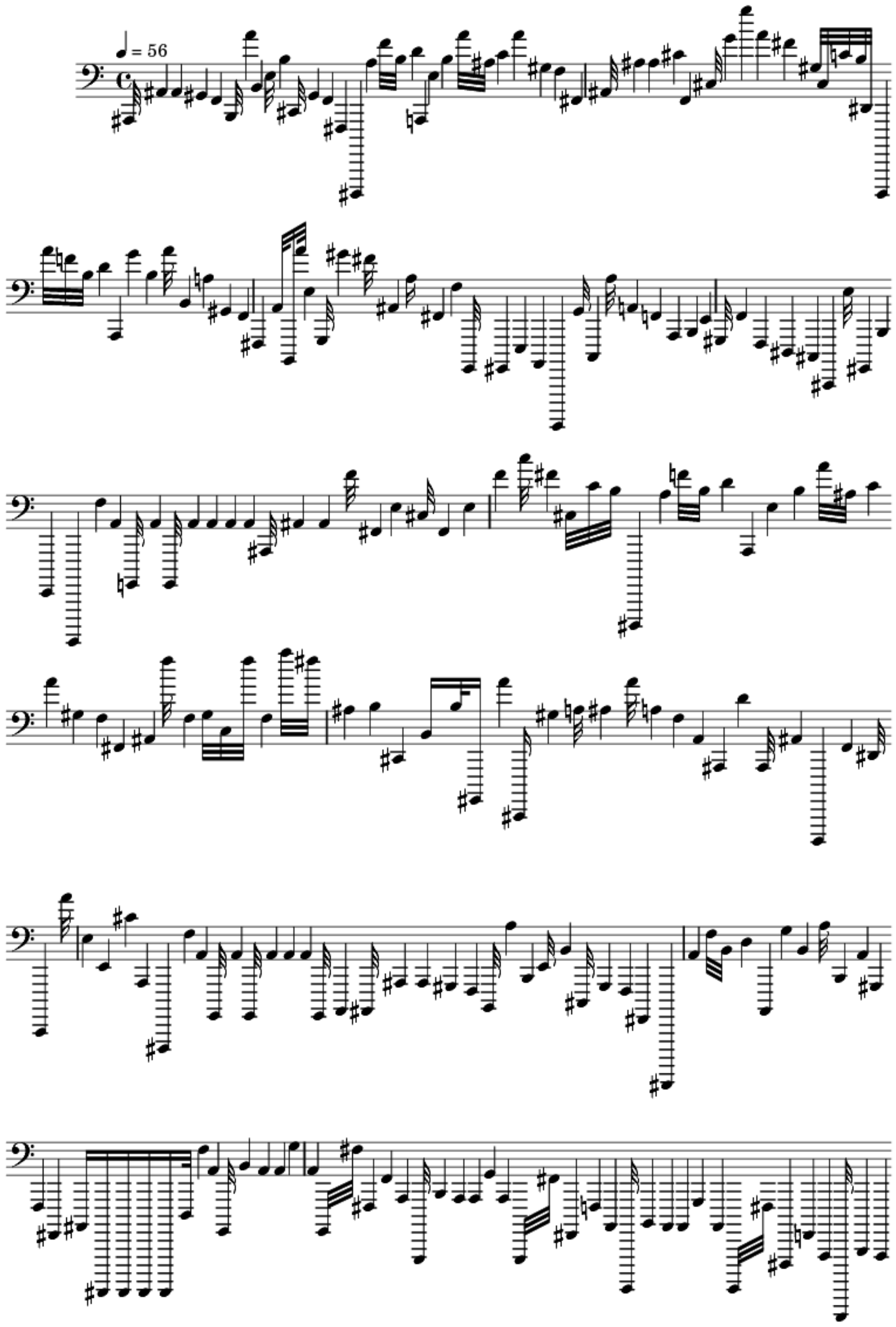


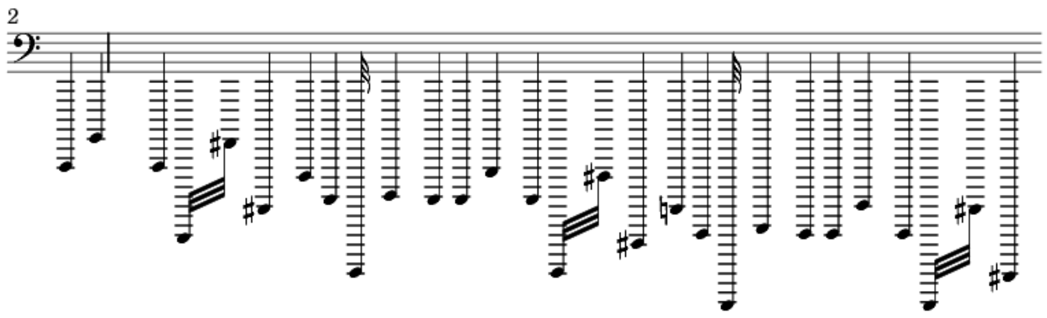
This piece was generated by sending the English lyrics of “Twinkle Twinkle, Little Star” to `Wordsongs`. The resulting output is repetitive yet varied, and reflects upon the repetitive and varied nature of lyrics. While still a bit atonal, this piece shows more of a tonal approach to composition than the previous example. Not surprisingly, the output resembles in no way the original Mozart tune, due to the approach taken by `Wordsongs` to translate strings to sound.

As the input contained a large amount of identical tokens, and `Wordsongs` is completely deterministic when it encounters a token, the output piece contained a large amount of repetition. This is a key factor in creating tonal music in the western tradition, and goes a long way towards helping the listener maintain a feeling of engagement with a certain piece. The variations keep the listener interested, while the repetitions reward them.

Musically speaking, this short piece demonstrates the very important principles of engagement and reward, through the use of repetition and variation. Experimentally, this piece demonstrates that with a well-crafted input file, `Wordsongs` can create interesting results that, though aurally may fall flat, at least express some interesting emergent compositional ideas in a way that can be controlled consciously by whatever agent is crafting the input.

3.1.2.3 HeyJude.txt.midi



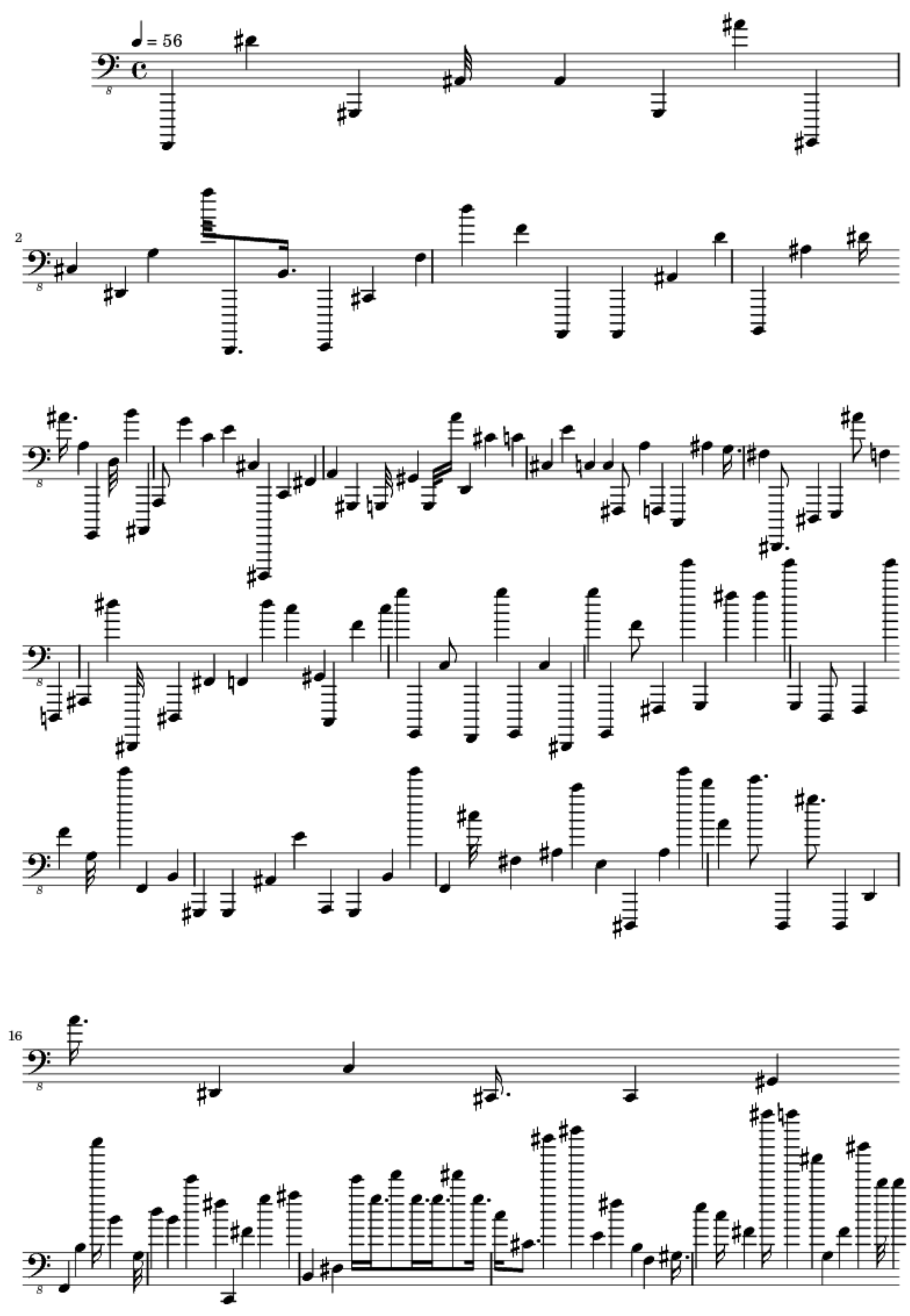


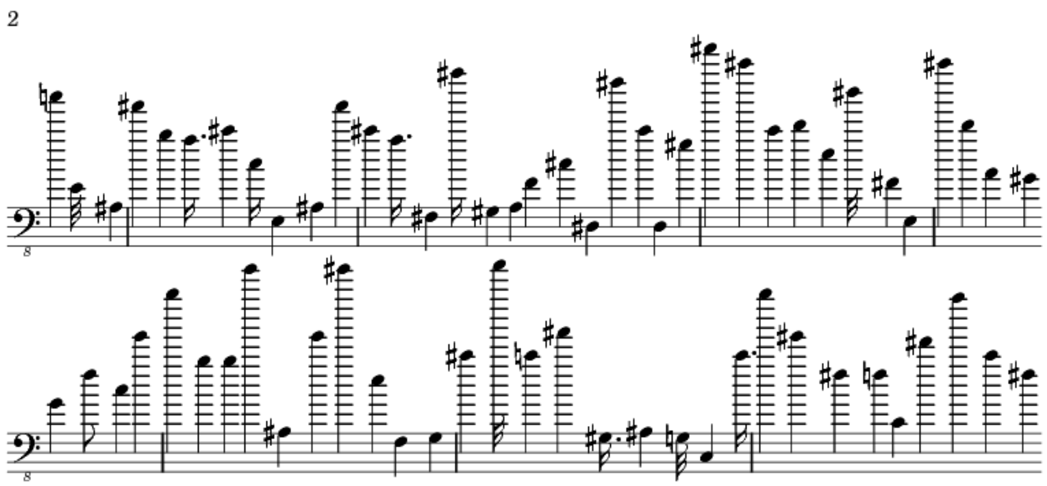
This piece was generated by sending the lyrics of The Beatles' "Hey Jude" through `Wordsongs`. Thought to combine repetition with variation in a suitable way, this input was chosen to create another example in the vein of the "Twinkle" piece above. This led to some interesting results, mirroring the "Twinkle" piece in style, but not in execution.

This piece has a larger amount of variation in the beginning, due to the highly varied verse content, when compared to "Twinkle." However, due to the end of the lyrics consisting of two repeating lines ("Na na na, na-na na na//Na-na na na, hey Jude") the end of this piece is highly repetitive as well, leading to a feeling of monotony. The contour of the tune is interesting from a musical perspective, as it surprisingly seems to apply harmonic thought in a reasonable manner, though this is largely luck due to the way `Wordsongs` processes tokens.

This piece shows the variation and repetition of the previous piece, along with a subtle highlight of harmony in intervallic relationships. Experimentally, this piece shows that `Wordsongs` is capable of producing harmony, but the process for creating this aspect of a piece depends largely on luck when defining the input, as it is difficult (though not impossible if the Java String `hashcode()` method is known) to determine the mapping of a particular token to its pitch by hand.

3.1.2.4 Google-Index-20100526.html.txt.midi


♩ = 56
2
16



This last example was composed by `Wordsongs` after looking at the HTML source code of the index page of `http://www.google.com` on May 26, 2010. It is a slowly moving piece, with a heavy concentration on the lower octaves. Though not strictly tonal, this piece exhibits the feeling of a composition in a minor key—lachrymose and longing for resolution to a major—and leaves the listener just a bit disappointed due to the abrupt ending.

The low pitches in this piece are really the meat of this, combined to a lesser extent with the long durations. The low pitches are a product mostly of luck, as the token-to-note conversion process is a bit abstruse to the end user as discussed above. The long durations, however, are a direct result of the length of the tokens. Google's source tends to be compressed to reduce bandwidth, and due to HTML not requiring much whitespace, the tokens tend to be extremely long strings of characters. This, combined with the constraints built into the program, created a slowly moving atmospheric rhythm for the piece.

Musically, this piece demonstrates the striving towards resolution of a relatively minor composition, along with the importance of rhythm and pitch. This experiment demonstrates the (albeit mostly random) ability of `Wordsongs` to create a piece that strives for tonality, and even attempts to mimic a key.

3.2 HexMusicExperiments

3.2.1 Description of HexMusicExperiments

`HexMusicExperiments` is a cellular automata-based approach to algorithmic composition. This system uses three different cellular automata running in parallel to generate octave, duration, and pitch groupings independently of one another, and is written in Java using JFugue. Running this program produces a musical output of loosely-defined chords following one another in sequence, with some overlap due to inconsistent durations, providing some interesting results.

The three cellular automata are of two different classes, and all of them follow a unique ruleset. Two of them, the octave and duration cellular automata, are one-dimensional automata belonging to the class that Stephen Wolfram classified under his binary rule system. The octave cellular automata follows "rule 90," which is defined by the following table:

current generation	XXX	XX-	X-X	X--	-XX	-X-	--X	---
next generation	-	X	-	X	X	-	X	-

Where X denotes a live cell, and - denotes a dead cell [41]. Along with its current state, each cell in the octave system stores an octave number it is assigned to, ranging between 3 and 7, octaves chosen so as to be close enough together for transitions between octaves to not be unnecessarily harsh. Each generation, after iteration, the set of live cells is collected, and the mode of the values of these cells is used as the octave for the resultant pitch grouping.

The durational cellular automaton works much the same way. This system follows "rule 30," which is defined as [40]:

current generation	XXX	XX-	X-X	X--	-XX	-X-	--X	---
next generation	-	-	-	X	X	X	X	-

As in the octave world, each cell in this cellular automaton stores a value corresponding to a duration, between a whole note and a $1/32$ note. Each generation, after iteration, the mode of the values of the set of live cells is taken and used as the duration for the resultant pitch grouping.

The third and final cellular automaton in use in `HexMusicExperiments` is one defined myself, titled "Peck's Rule." This is a hexagonal two-dimensional cellular automaton which follows rules similar to Conway's Game of Life. In "Peck's Rule," any live cell stays alive in the next generation if it has two or three live neighbors, and a dead cell comes alive in the next generation if it has one, two, or three live neighbors, with all other cells dying or remaining dead. This rule can be classified as the life-like rule `S23B123` in a hexagonal world.

Each cell in this cellular automaton stores a value corresponding to a pitch class, which is entered by the user in the form of a seven-pitch scale. Each generation, after iteration, the set of living cells is created for the first row, and the mode is taken of this set to generate a pitch, and adds it to a set of pitches (the pitch grouping referred to above). This process continues for each additional row in the world, where any duplicate pitches are simply omitted from the resultant set.

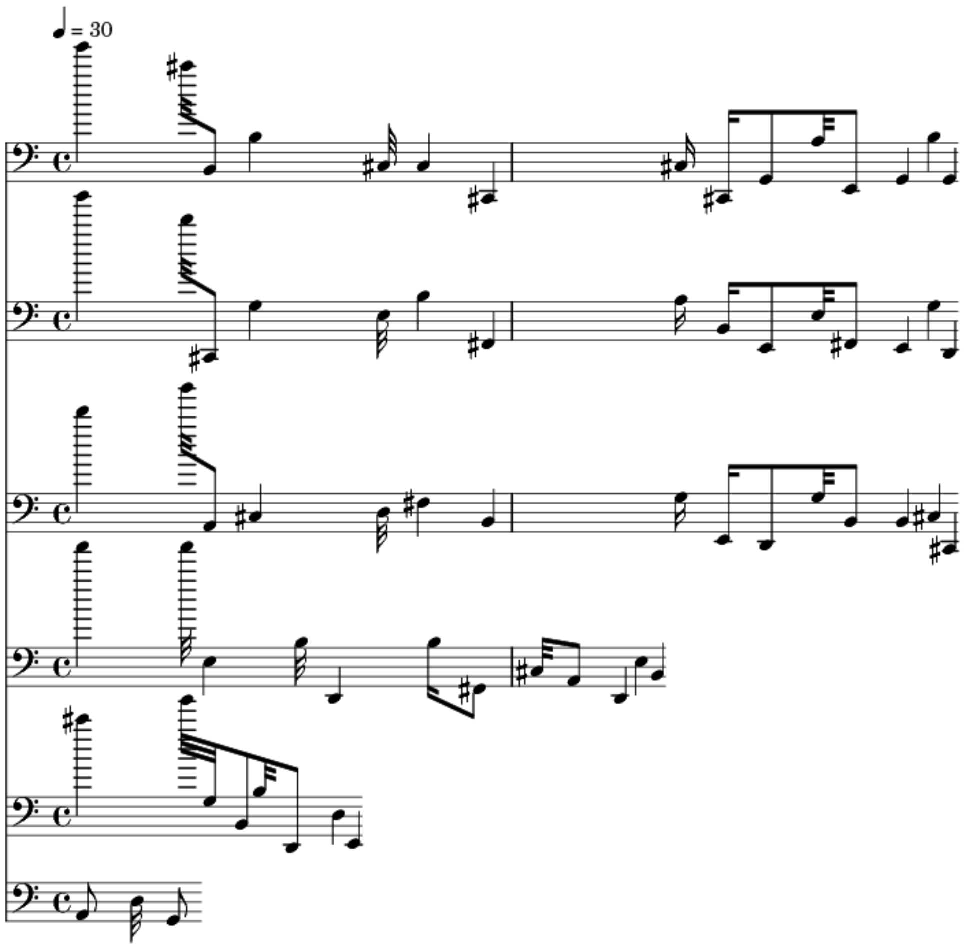
After all three pieces of data are collected, they are concatenated to the end of the current piece as a set of concurrent voices. Due to the way MIDI handles voices with unequal duration, oftentimes overlapping chords are formed, generating interesting patterns which lead the piece from multitudinous voices in the beginning to only a few in the end.

When `HexMusicExperiements` is run, the user is prompted for a number of initial parameters. First, the size of the cellular automaton which defines the octave is established, followed by the size of the durational cellular automaton, and the height and width of the pitch grouping cellular automaton. After this, a seven-note scale is requested, in the form of a string of space-delimited JFugue tokens [21].

Upon providing these parameters, the system iterates for the first time (using a random initial distribution of live and dead cells for each cellular automata), displays the proper output, and provides the user a chance to end the system, or continue iterating. A press of the enter key tells the system to iterate once more, applying its results to the end of the current piece, and typing `quit` tells the system to play the piece, save it as a MIDI file, and exit.

3.2.2 Pieces composed using HexMusicExperiments

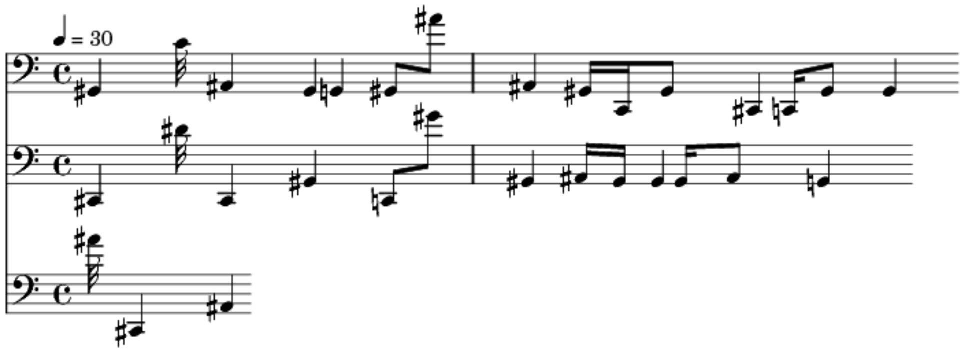
3.2.2.1 dipper-Bm.midi



This piece was generated by running `HexMusicExperiments` with rather large parameters for fifteen generations, in an attempt to create a piece that sounded "full" due to the number of simultaneous voices. The resulting output is chaotic at times, but due to using only notes in the key of *B*-minor, remains coherent to the listener enough to prevent this piece resembling a cacophony of disjoint pitches.

The jarring changes in chord structure apparent in such a short time frame are evocative of the works of Xenakis. Though rather disparate and patchy, often there is a sense of resolution after a particularly disjoint chord. Musically this short piece is rather obtuse, but it allows for study of the process taken by `HexMusicExperiments`.

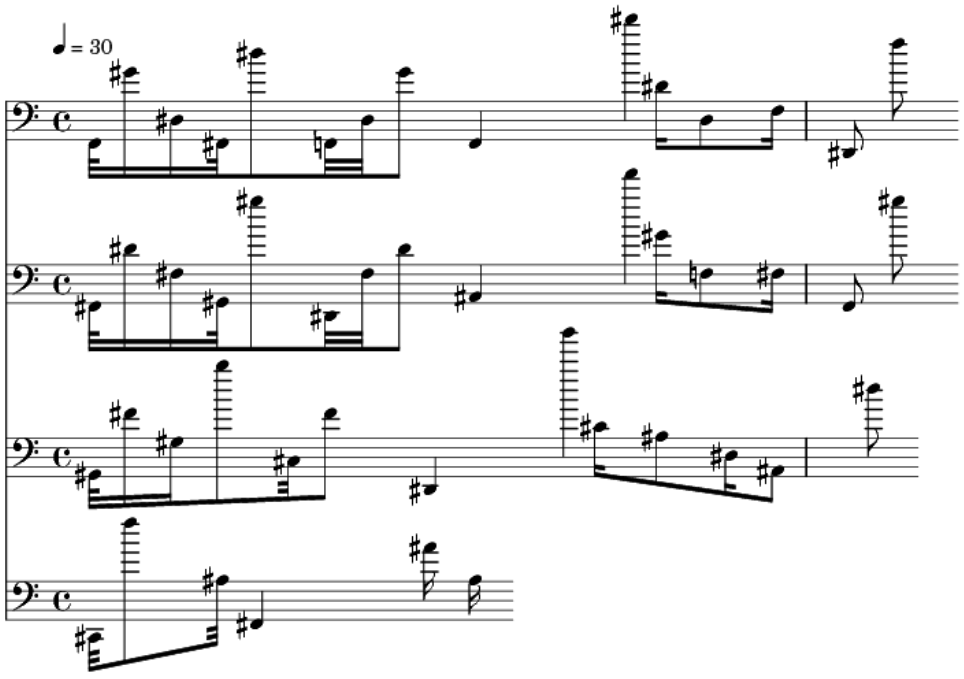
3.2.2.2 dipper-A♭.midi



Taking the opposite approach to the previous piece, `HexMusicExperiments` was run with small parameters for fifteen generations in attempts to create a “sparse” sounding piece. This attempt was rather successful, as oftentimes there is a feeling that there should be another voice along with the leading lines. Despite sticking to the key of *A*♭-major, the piece has a sort of depressed feeling, as if something is missing.

The sparseness of the piece, combined with the relatively long durations, creates a haunting feeling. Musically, this piece demonstrates the ability of pitch and duration to have an effect on the listener’s perceptions. Experimentally, this shows that with the proper parameters, `HexMusicExperiments` may produce some interesting results, despite the high amount of luck involved.

3.2.2.3 dipper-D♯m.midi



This piece composed by `HexMusicExperiments` was created by taking a sort of middle-ground approach between the previous two examples. Running `HexMusicExperiments` with medium weighted parameters for fifteen generations produced this piece in the key of $D\sharp$-minor. A bit stark in its execution, the chords seem to be having a standoff with each other for dominance of the piece, and as time progresses, they seem to fracture away into solitary notes, standing on their own.

This fracturing provides an interesting effect musically, with the listener attempting to grasp onto these now disparate remnants of what was previously a strong chord. There is resolution, but overall the piece feels unresolved. Experimentally, `HexMusicExperiments` performed rather well in this example, providing a piece that isn't too crowded sonically, and that isn't under-voiced either.

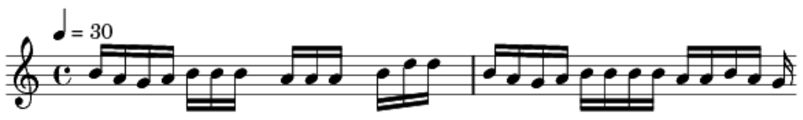
3.3 Markov Machine

3.3.1 Description of Markov Machine

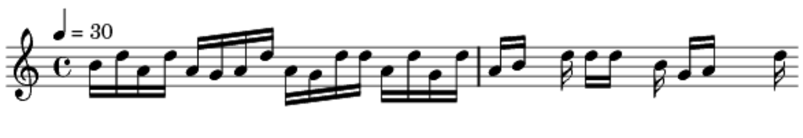
`Markov Machine` is a simple program that attempts to replicate the style of a selected work by performing Markov analysis upon the input. After reading the input piece, `Markov Machine` performs a zeroth-degree Markov analysis upon the input, producing an output piece of equal length but with notes chosen effectively randomly from the set of notes in the input. Following this, a first-degree Markov analysis is performed upon the input, producing a piece that is more in line with the musical style of the input. This quick program was designed as an introduction to Markov processes and also to show that such an approach is a viable option in the school of style-replicationist algorithmic composition.

3.3.2 Pieces composed using Markov Machine

3.3.2.1 Mary Had a Little Lamb

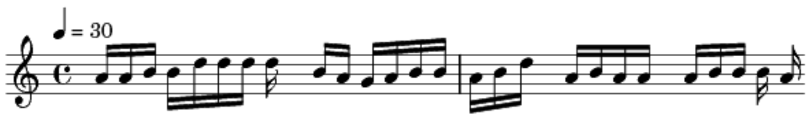


`Markov Machine` was fed the notes of “Mary Had a Little Lamb” to work on. The score above is the original piece. Once analyzed, `Markov Machine` produced a zeroth-degree Markov analysis, producing a piece equal in length to the original, consisting solely of notes from the original, with no weighting applied. The following is the result:



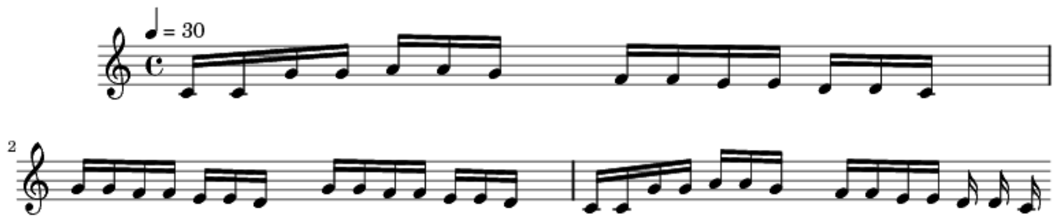
An interesting piece bearing no resemblance to the original, this small melody provides the listener with some sense of direction, despite being essentially random. The repeated *D* notes are of interest, considering that of the entire original piece, there are only two, yet they form a prominent role in this reworking.

After the zeroth-degree analysis, a first-degree Markov analysis was performed upon the original piece, producing the following:

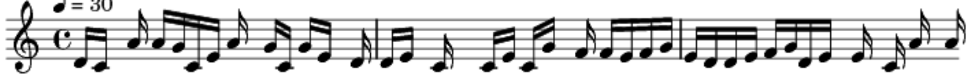


This piece, which even contains quotes of the input, is a testament to the ability of Markov chains to emulate styles. Given the limitations of `Markov Machine` (namely, only one piece in the corpus and only first-degree analysis), this piece shows the possibility for future expansion upon the idea of Markov-assisted music composition.

3.3.2.2 Twinkle Twinkle, Little Star

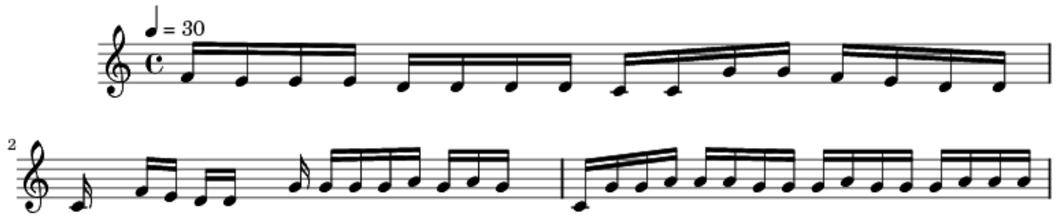


`Markov Machine` was run with "Twinkle Twinkle, Little Star" as its corpus, shown above. Analyzing this, `Markov Machine` generated the following zeroth-degree reworking:



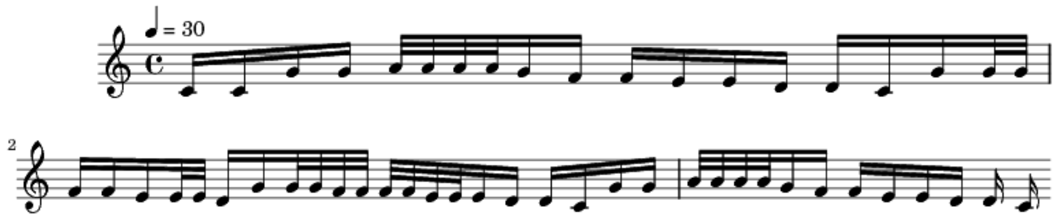
Again, an interesting diversion, but with little in resemblance to the original piece. This piece is actually more interesting the the first-degree result obtained below, because it contains variation and subtle motives, even being completely left to chance.

The first-degree result of this run is as follows:



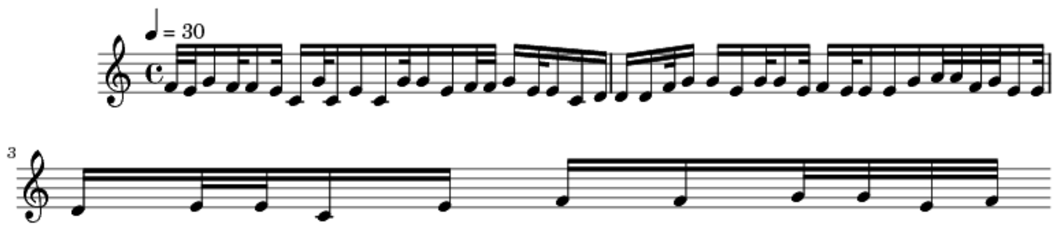
This piece highlights and concentrates on the weaknesses of `Markov Machine`. Due to the rigid regularity and repeated note structure of the original piece, even a first-degree Markov reworking of "Twinkle" leaves much to be desired. This result has little variation, and is comprised of oftentimes painful groups of repeated notes that terminate abruptly, only to lead into another such group.

3.3.2.3 Baa-Baa Blacksheep



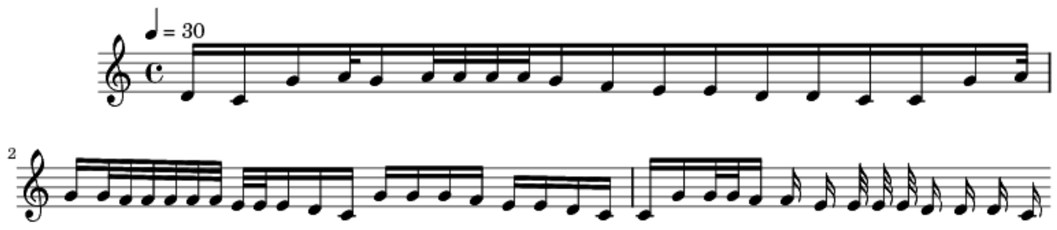
Feeding `Markov Machine` "Baa-Baa Blacksheep" (above) was done to test the program's performance with variable note durations. Due to `Markov Machine`'s implementation, a note is defined as the combination of pitch and duration, so an *A* quarter note, for example, would be a different note than an *A* eighth.

Upon analysis, `Markov Machine` produced the following zeroth-degree reimagining of the original:



No noticeable connection between this piece and the corpus exists, aside from the shared set of notes. However, some interesting arpeggios are to be found, and the whole piece feels sort of whimsical.

The first-order analysis of the corpus produced the following:



This test proved that `Markov Machine` can operate reasonably well with notes of different duration, as results were similar to those of the "Mary Had a Little Lamb" run above. Containing quotes of the original piece, and retaining an overall feel of the corpus, `Markov Machine`'s version of "Baa-Baa Blacksheep" demonstrates again the utility of Markov analysis in style replication efforts.

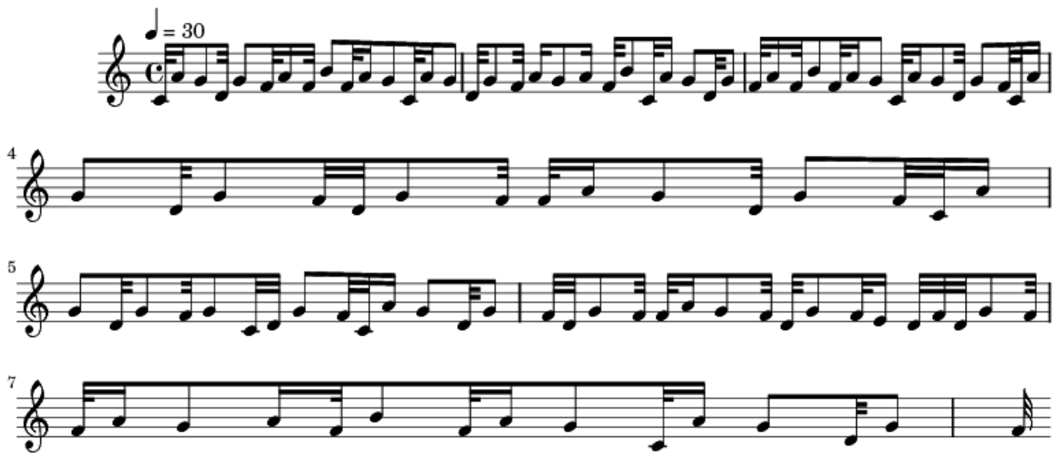
3.4 LCompose

3.4.1 Description of LCompose

`LCompose` is a program that assists in writing music with the use of L-Systems. When the program starts, it prompts the user for an alphabet in the form of a space-delimited string of JFugue tokens, a rewrite rule for every element in the given alphabet, and a seed (another name for the axiom) for the system to operate under. Using these, the program simply iterates the system and plays the current data string in between each iteration, pausing to prompt whether to iterate again or stop and save as a MIDI file. `LCompose` was the topic of a talk I gave at the QUEST symposium at SUNY Oswego in April 2011 [30].

3.4.2 Pieces composed using LCompose

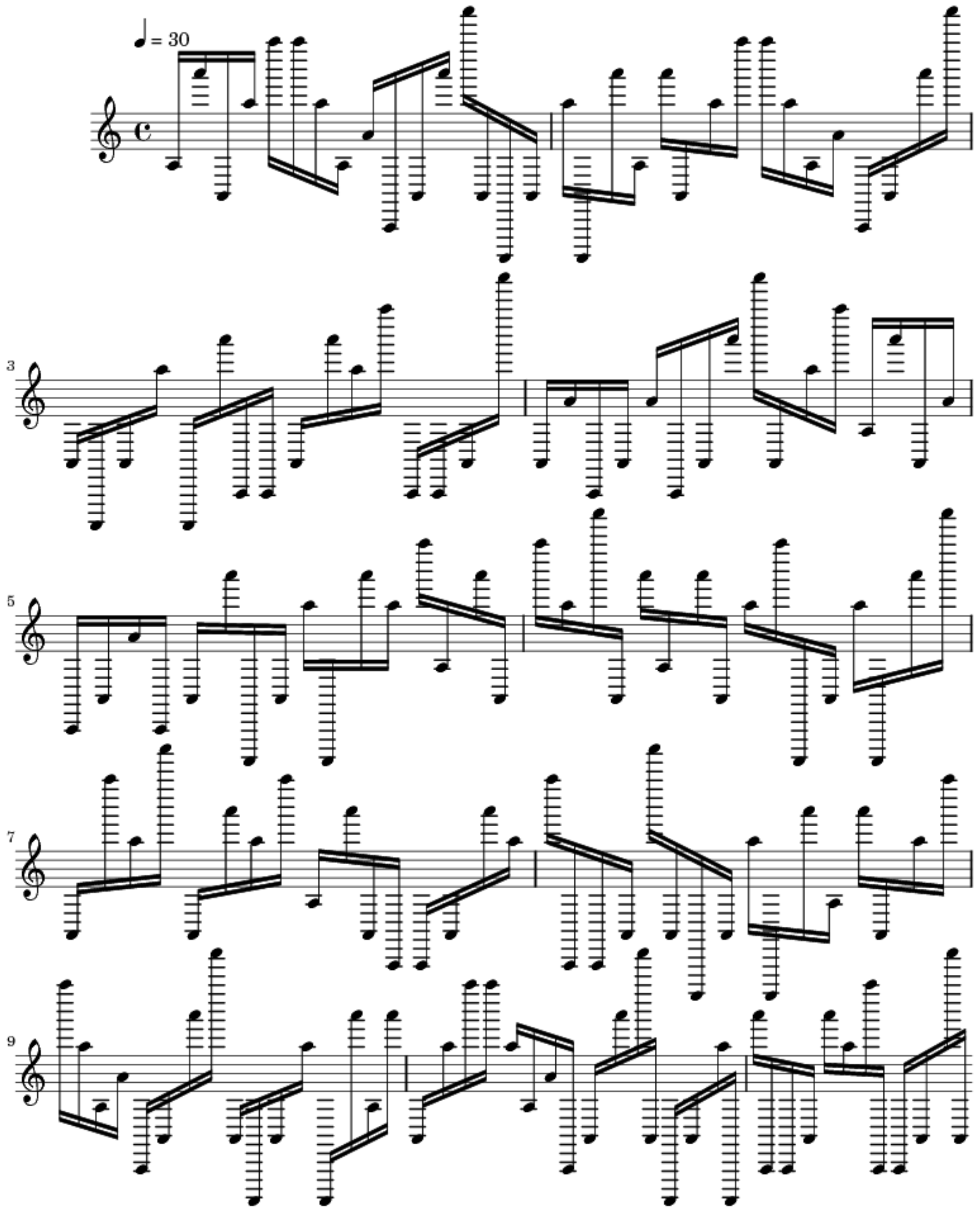
3.4.2.1 good-example-1.midi



This first example was composed by `LCompose` by inputting a relatively simple collection of symbols—the C-major scale with differing durations—and entering essentially random rewrite rules. No thought was put into the musicality of the rewrite rules; the only restriction on them was that they be sufficiently varied and contain between one and three symbols. In sticking to the C-major scale, an attempt was made at obtaining some sort of coherence out of the piece.

This piece surpassed my expectations. It contains repeating parts, a signature of L-System composition, as well as being surprisingly musical and pleasing. Though a bit repetitive, the piece maintains an overall feel of uplifting melodic moments interspersed with subtle variations in duration and contour. The intervals between the notes turned out far better than could have been hoped, creating a harmonious, consistent trek through the piece.

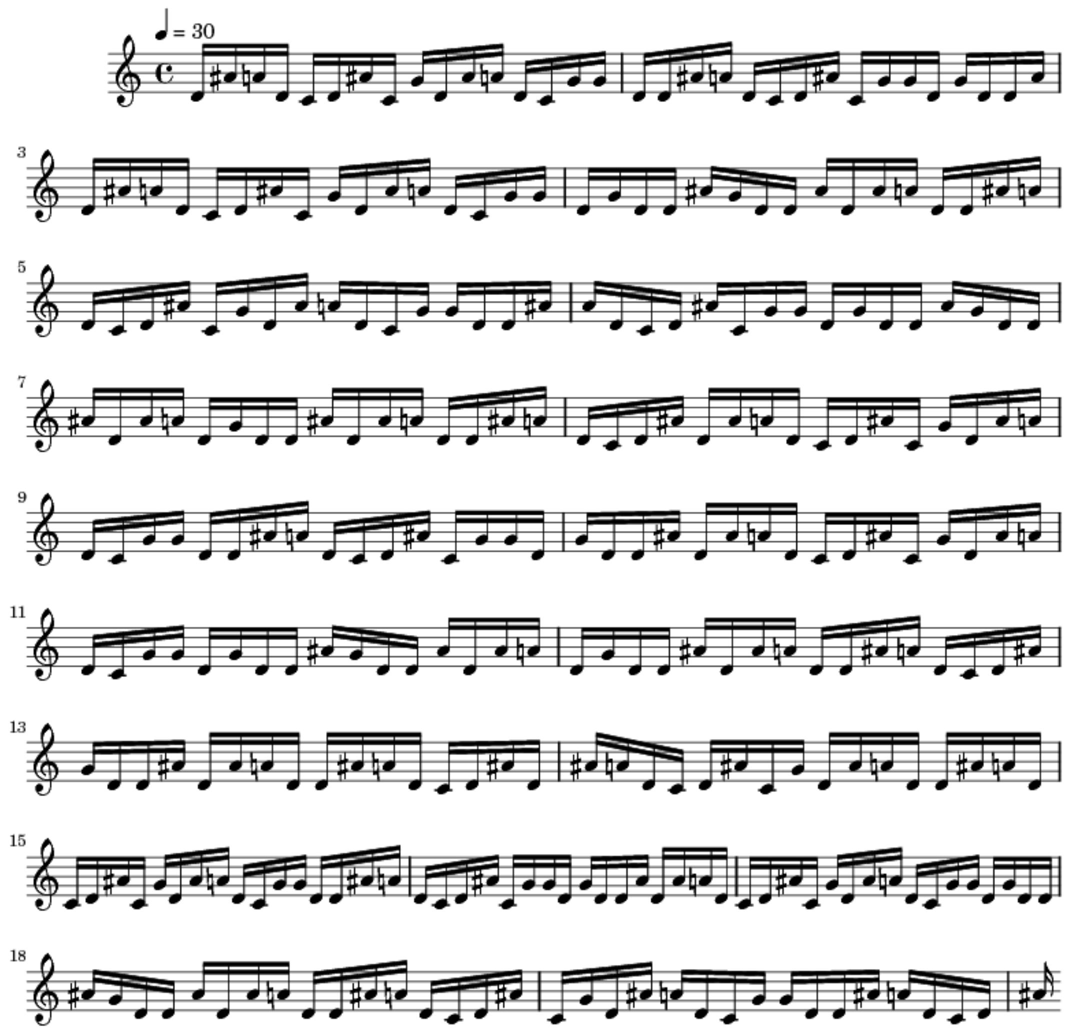
3.4.2.2 pitch-class-A.midi



This is an experimental piece composed solely of notes in the pitch class of *A*, in nine different octaves, to abolish the concept of a key—or rather, creating an even-tempered one-note scale. There is no variation in duration, leaving absent entirely a sense of rhythm. As can be expected, the score above is extremely chaotic, but the piece itself when sonified leaves the listener with a feeling of balance and order.

Whether or not this could be considered music in the western sense, this piece demonstrates some very powerful capabilities of L-Systems, and by extension `LCompose`. Firstly, the capability to operate on completely generic symbols (in this case, `A1`, `A2`, `A3`, etc.) is useful when trying to isolate a single element of a much more complex system, here used to isolate pitch-class at the exclusion of rhythm and scale. Second, repetition with subtle variation allows for very stable output sequences with just the right proportion of noise in the signal. Both of these capabilities may be used to great extent in modernist music, particularly in minimalist works.

3.4.2.3 missymorbid.midi



This piece was created by a friend using `LCompose`. She entered the *G*-minor scale, all in equal durations, and assigned rewrite rules according to traditional intervallic patterns. The result is an evocative, haunting piece that feels not at all computer-generated, save for the lack of rhythm. The repeating triads are the main motif of this piece, and provide a balance and orientating force for the listener.

Displaying what `LCompose` can do in the hands of someone who had never used a computer to generate music before shows the simplicity and power of algorithmic composition when applied correctly.

3.5 ALM - Artificial Life Music

3.5.1 Description of ALM

`ALM`, short for Artificial Life Music, is an algorithmic composition approach using cellular automata, L-Systems, and genetic algorithms. Written in Common Lisp, with an auxiliary program written in Java with JFugue, this program demonstrates the loose coupling of components that Andrea Valle finds imperative for useful algorithmic composition systems [36].

When the program starts up, the user enters an initial alphabet from which a number of different L-Systems are generated. These L-Systems then form the initial population for a genetic algorithm. The genetic algorithm utilizes a fitness function to evolve these initial L-Systems into a population of more fit L-Systems, which are then assigned on a one-to-one basis to cells in a one-dimensional, toroidal cellular automata. This cellular automata is used as a selection grid, with each cell maintaining a history of its previous states to act as a weighting factor. A seed, from the alphabet entered previously, is chosen at random, and the cellular automata is iterated. The cell with the largest weight—the number of generations in the cell's history that the cell was alive—is chosen as the representative of the L-System to be used. The L-System attached to this cell is used in rewriting the current composition, and the output is stored as the new composition. The program continues in this fashion, using the Java daemon to sonify the examples, until the user decides to terminate the program, at which point the composition is saved as a MIDI file.

The cellular automata used in this system is defined arbitrarily to be "rule 86," defined by:

current generation	XXX	XX-	X-X	X--	-XX	-X-	--X	---
next generation	-	X	-	X	-	X	X	-

The cellular automata begins with a random distribution of live and dead cells, has a toroidal grid, and each cell begins with a history containing just its initial state. Every iteration, the cell's new state is concatenated to the end of this list, and when a weight value is needed, a simple count of all occurrences of X is taken.

The L-Systems in use are standard, deterministic L-Systems. However, they are evolved out of an initial population of randomly generated L-Systems, striving towards fitness. The fitness metric in use in this project was designed to attempt to capture the even-balanced nature of notes in serialist music. This particular fitness function fails to capture the idea of the twelve tone row, but it was not designed to. The only aspect of serialist music that the fitness function attempts to account for is the even counts of every note in the scale, and it does this by running the system (with a random seed from the alphabet) for six generations (arbitrarily chosen), and performing the following process on the output:

- Count the occurrences of each symbol from the alphabet in the output. Assign these numbers to values a, b, ..., y, z.
- Add every possible ratio of two of the above values together, *ex.* $\frac{a}{b}$, $\frac{a}{z}$, *etc.*
- Divide that value by $n \times (n-1)$, where n is the number of symbols in the alphabet.
- Take the reciprocal of that value, and constrain it to the range $[0, 1]$.

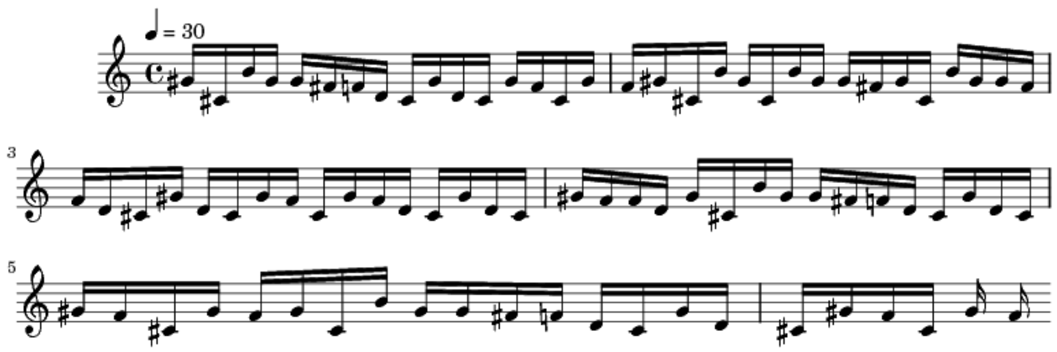
This ends up with a decimal value between zero and one, representative of how well-balanced the output is in terms of note distribution. The formula for this is as follows:

$$f(x) = \frac{1}{\left[\frac{\left[\frac{a}{b}+\frac{b}{a}+...+\frac{y}{z}+\frac{z}{y}\right]}{n\times(n-1)}\right]}, min0, max1.$$

where $f(x)$ is the fitness of L-System x [29].

3.5.2 Pieces composed using ALM

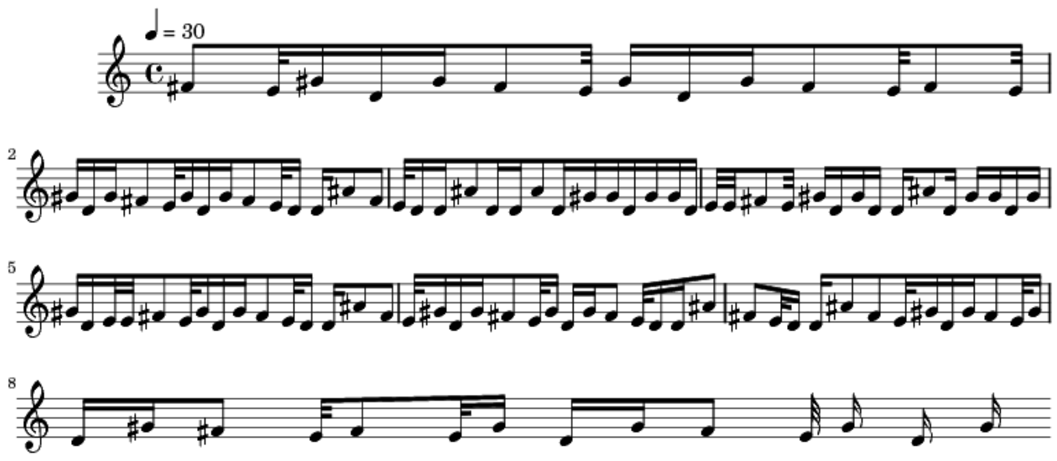
3.5.2.1 LifeInAMajor.midi



This piece was composed by `ALM` in the key of A-major. A medium-range desired fitness level (`0.86`) was selected to allow for slight variations in the rewrite rules, and a population size of 100 was specified, to allow for more relaxed competition in the evolution process. After eleven generations, a population with an average fitness meeting the desired fitness level was found (`0.9363232`), which was then used to compose a small piece through six generations of the evolved L-Systems.

The result is relatively melodic, given the constraints. A small quote from "Fiddler on the Roof" appears early and often in this piece, and really evokes the idea of a traditional, toiled life. However, the simple theme suffers from far too much repetition, and combined with the complete lack of rhythm—all notes are of equal duration—overall this piece falls flat.

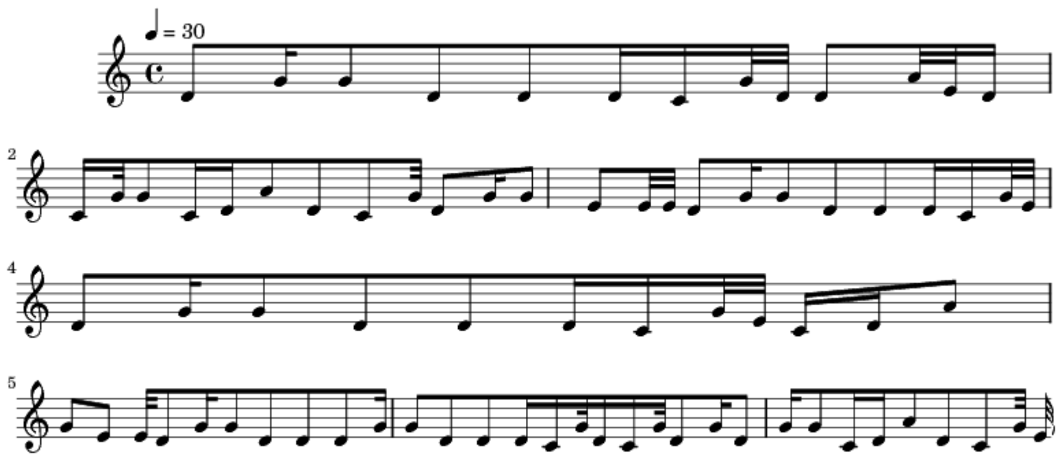
3.5.2.2 MovementsInCWholetone.midi



This piece was composed in the C-wholetone scale, with differing durations. A slightly-lower desired fitness level (`0.84`) was selected than in the previous example, to allow for more variation, along with a smaller population size (75), to increase competitiveness in the evolution process. After only four generations, a suitable population was found (average fitness of `0.9143303`) from which six generations was processed into music.

The resulting piece is interesting in its repetition, almost evoking a Philip Glass-like sense of 'static movement.' The variations in duration lend a nice rhythm to the piece that allows one to feel as though the music is making progress, and also results in a very strong expectation-reward experience. However, there are violent, often unsettling changes when one is expecting the repetition of a particular part, in which this expectation-reward experience is violated. This lends an almost Schoenberg-like feel to the piece as well, violating the listener's desires for resolve. This piece stands as an interesting exercise in exerting atonal thought into a tonal piece, even if completely generated by computer.

3.5.2.3 BluesInE.midi



This piece was composed by giving `ALM` the blues pentatonic scale in *E*, with each pitch having three different durations. A medium-range desired fitness level (`0.86`) was chosen once again, and an even more constricted population size of 50 to increase competition to a savage degree. Seven generations of evolution took place to determine a suitable population (average fitness of `0.93914425`). Six iterations of this population's L-Systems produced the result.

Though not immanently bluesy, this piece does contain some interesting elements. The irregular rhythm of this piece is the major factor, creating an uneasy, jerky feeling. Due to the nature of L-Systems, combined with chance, the strange pattern of durations above is exhibited. Though each L-System is clearly defined in its own right, the implementation of `ALM` lends much to chance, creating an interesting experience for both composer and listener. A weakness, in many regards, that leads to pieces such as this with no clear place as anything other than an interesting diversion.

4 Discussion

4.1 Discussion on aesthetic

The aesthetic of music composed with the assistance of algorithmic process varies wildly. As demonstrated by Cope's `EMI`, extremely fine-tuned computer programs designed to compose algorithmically are very capable of producing music that is as aesthetically pleasing as that of the great masters [10]. Coupled with the fact that several well-known composers produce music almost exclusively algorithmically (Xenakis, etc.), or apply algorithmic thought to their traditional compositional efforts (Cage, etc.), it can be shown that algorithmic composition is most certainly a viable approach to music writing [11, 13].

Many times however, algorithmic composition and the works it engenders are exercises in theory over practice, with aesthetic falling just short of the intended goal. Certainly this is the case in the works presented in the previous section, where an exploratory approach to algorithmic composition was taken with little regard to aesthetic in comparison to hard theory. Oftentimes the results received from running the above programs are aesthetically lack-luster, or even abhorrent in the ears of the right (or wrong) listener, but provide great insight into the process and thought behind the approach.

In my own subjective opinion, of the approaches provided `ALM` produced the least aesthetically pleasing results. But this is of no matter when studying the process that lead to the results. The output of `ALM` demonstrated the inherent randomness of the entire approach, and yet also managed to capture the sense of evolution and structure inherent in genetic algorithms and L-Systems in the regularity of the pieces, both in pitch distribution due to the fitness function, and in repetition due to the rigid rewrite rules of the L-Systems involved.

When speaking of aesthetically pleasing results, look towards `LCompose`. The pieces produced by `LCompose` seem to hold up the best against the test of human enjoyment, and a large part of this is due to the direct involvement the user has with the compositional process. `LCompose` does almost none of the musical work itself—that is all elaborated by the user—and the resulting pieces are likewise directly influenced by the user's tastes and decisions. In analyzing their composition, the repetition is gathered from the forms produced by the rewrite rules imposed by the user, which in turn allows a (slightly opaque) look into the thoughts of the composing user.

4.2 Complications encountered

During the research and programming for this exploration, a few complications were encountered. Through more research and clever manipulation, some of these complications were mitigated. A few, however, remain in the final product.

Throughout the course of this research, perhaps the largest complication encountered was crafting the fitness function for `ALM`. Devising a fitness function is a crucial part of ensuring any genetic algorithm performs in a way that positively refines solutions to match a goal scenario, yet seems to be the part that is hardest to grasp when designing the system in which it will operate. In the end the fitness function presented in `ALM` is far too sensitive to even the smallest mutation in a gene, rendering its results wildly unpredictable. For an example of these results, consult the following table of a run of twenty generations with 50% copy, 49% crossover, and 1% mutation rates [29]:

Generation	**Fitness**
Initial fitness	0.28499436
1	0.5026495
2	0.6484151
3	0.6143918
4	0.36710966
5	0.64340574
6	0.7197356
7	0.602708
8	0.8284424
9	0.6633316
10	0.6908027
11	*0.7634973*
12	*0.23890796*
13	0.5888962
14	0.7547191
15	0.5993262
16	0.7726623
17	0.84089243
18	0.7832393
19	0.69761515
20	0.6627491

Particularly astonishing is the massive degradation in fitness between populations 11 and 12. However, in performing a run of twenty generations with 50% copy, 50% crossover, and 0% mutation rates, the following results are obtained [29]:

Generation	**Fitness**
Initial fitness	0.26688015
1	0.47937027
2	0.63973
3	0.7401396
4	0.9233875
5	0.95210725
6	0.989678
7	0.996584
8	0.996584
And so on...	0.996584

Here we see that the population approaches stability incredibly quickly, and once stabilized refuses to change even the slightest bit. So, in order to make this fitness function work, a relatively vital part of the genetic algorithm had to be left out of the equation to allow for the fitness function's hypersensitivity to not adversely affect performance—albeit at the cost of stagnation and variation.

Another complication encountered was how to internally represent Markov chains in Java programs. Ignoring the fact that `Markov Machine` was essentially "thrown together" to demonstrate an idea, the programming of this tool was approached haphazardly at best, and is thoroughly in need of reorganization and refactoring with regards to data structures. The main reason the program seems incomplete and only models up to first-degree Markov analysis is due to a poor choice in design due to simply wanting to use the tool as soon as possible. In this regard, it is the least mature tool of the ones discussed above, and the one that could most benefit from a rewrite.

Early versions of `LCompose` suffered from a terribly unintuitive command line, using non-standard symbols to represent rewrites. In cleaning the program up for a presentation at the QUEST symposium, these issues were addressed, and as a by-product of these efforts `LCompose` changed from a declarative knowledge model of representation to an inquiry based model whereby the program prompts the user for input [30].

The rules for determining the alignment of cell values in `HexMusicExperiments` are complex, and after many hours of attempting to discover a way to generate them algorithmically, eventually a pencil and a piece of paper were used to figure them out by hand and hard-code them into the program. Unfortunately, this leads to inflexibility in the system, but as `HexMusicExperiments` is a tool intended for personal use and not widespread circulation, this seems to be a minor problem.

One major complication that remains unsolved in all of the tools presented is the complete and total lack of any error checking. In attempts to pull ideas out and place them into cognitive tools, the importance of error checking faded into obscurity. In many cases error checking code would be a trivial task to implement, and would be a good exercise in validation techniques.

4.3 Areas for further study

Whenever research is done into a topic, new areas for study reveal themselves. The research put into this project was no different.

In researching algorithmic composition, many interesting approaches and techniques are discovered and discussed that allow room for future innovations. In particular, Gerhard Nierhaus' work illuminates several of these approaches and offers advice on following the threads spun by previous researchers [28]. Several topics in Nierhaus' work are intriguing, and are very fit for future exploration.

Through `LCompose` and `ALM`, L-Systems for use in compositional systems were approached and examined. L-Systems are only one small part of an entire family of generative grammars, however, and these systems have extensive use in algorithmic composition [28]. One extremely interesting approach to algorithmic composition involves the use of context-sensitive grammars to selectively piece together different parts of a larger musical piece. It is easy to imagine how, using a type of L-System that is capable of understanding its environment, a much more coherent and varied piece of music may be composed. Or, climbing one step above the context-sensitive grammar world, introducing finite state machines or push-down automata to the world of algorithmic composition could provide even more compositional power, however at a cost of increased complexity. It is not a stretch to say that such systems could easily compose entire symphonies, given the correct fine tuning by the human crafter.

Transition networks are another interesting topic. Though related to Markov chains, transition networks are different in implementation and often different in goal [28]. Transition networks, extended into the realm of artificial intelligence to create artificial neural networks, offer some serious competition for musical composers. With the correct training set and a powerful enough machine, an artificial neural network can theoretically model the human mind. Though far from this in practice, astounding breakthroughs have been made in various areas, including musical thought and composition [28]. When the study of neural nets grows to a point where a reasonable model of the human mind may be made, computer-composed music may very well capture emotion and feeling that seems to be lacking in most computer-composed pieces.

Chaos theory, stating that a small change in one part of a system affects an enormous change in another part of that system at some future point in time [28], has concrete proof in the following L-System example: given the axiom `A` and the rules `{A` $\rightarrow$ `AB, B` $\rightarrow$ `A}`, the fourth generation string returned is `ABAABABA`. But simply replace the rule `B` $\rightarrow$ `A` with `B` $\rightarrow$ `B` and the string `ABBBB` is returned on the fourth generation, showing how a small change early on leads to drastically different results later on down the line. In musical composition, this theory has interesting implications, and may be seen in a few examples by other researchers [28].

This research has also shown promise even outside of algorithmic composition related efforts. By allowing me to attain an understanding of cellular automata systems (and in particular Conway's Game of Life), a personal project in which social dependency and survival are simulated has been initiated by way of a heavily modified version of Conway's Game of Life interspersed with concepts in network science [31]. Through connections with cells outside of their immediate neighborhood, some cells may attempt to lengthen their lifespan by calling upon their "friends" in times of need (*i.e.* when the cell would have normally have died due to isolation) [31]. To my knowledge, few if any prior attempts at a model of this sort have been attempted, and this branch of research has also influenced some interesting thought in composing "societal music" by interpreting the results of this model musically in some fashion.

As a side note, most of the techniques used in algorithmic composition have also been used to varying degrees in the creation of algorithmic art. Fractal art, for example, is well known and applies the simple principle of self-similarity, and many fractals may even be modeled by L-Systems [28].

Overall, algorithmic composition is a field which is far too large to fully study, but also one that allows for exploration in ways unimagined previously.

5 Conclusion

5.1 Summarization

Algorithmic composition is an interesting topic that has garnered increased attention in the age of computing, despite its medieval roots. From the work of Guido d'Arezzo to the research of today, algorithmic composition offers intriguing puzzles and increased insights into musical and algorithmic thought. Covering the realm of algorithms rather extensively, algorithmic composition as a research topic also allows one to become acquainted with many extremely useful formalisms.

In applying these formalisms and studying them in-depth, one finds many opportunities to meld musical thought into the process. *Wordsongs*, written as a simple stepping stone into algorithmic composition, has created some of the more interesting pieces discussed in this paper. Through subtle manipulation, *Wordsongs* evolved from chaotic noise maker to semi-organized composer. Moving on from *Wordsongs*, `HexMusicExperiments` was a first step into an accepted formalism-based compositional practice. By applying cellular automata in differing forms and topographies, a simple chordal generator was created. Though the results were not always stellar, this application provoked thought and insight into the process of cellular automata music.

`Markov Machine` built upon the concept of Markov chains, and in so doing was able to produce relatively well-formed examples of pieces "in the style of" several children's songs. Chosen for their inherent simplicity and repeating form, these examples have shown that even a first-degree Markov analysis shows promise in compositional efforts.

`LCompose`, written as an introduction to L-Systems, allows for more direct manipulation of the results than in the prior examples. By explicitly defining the rewrite rules of the system, `LCompose` mirrors extremely closely the user's intent in creating musical output. However, due to the rigid nature of L-Systems, `LCompose` often falls short of composing an interesting work that lasts more than a handful of bars.

`ALM` allows for variation in the L-Systems-based approach that isn't offered by `LCompose`, but at the price of both increased complexity and decreased user-interaction. `ALM`, though a reasonably thought out system, relies upon a single point of failure in the form of the fitness function that tends to cause unimpressive and turbulent results.

In hindsight, all of these approaches could be improved in some way, with the most common fix involving error-checking. These programs are a bit fiddly to use, due to the lack of input verification, and as such may cause unease in the end user. As an exploratory tool these have allowed for some interesting insights and have worked well enough to produce the examples above, among many others, but as a program for any end-user aside from the developer they may very well fall flat.

In researching and programming these algorithmic composition tools, exploration in fields outside of algorithmic composition have been opened, allowing for even more future research. Algorithmic composition is an incredibly diverse interdisciplinary study that combines the best parts of mathematics, computer science, cognitive science, music theory, automaton theory, linguistics, philosophy, and psychology into a rewarding pursuit for knowledge and understanding.

5.2 Looking forward

The future of algorithmic composition is unclear, however many new names are showing up and making waves in the field. With new and continuing research by the likes of Eduardo Reck Miranda, David Cope, and Michael Edwards, more light is being shed on the concepts of algorithmic composition, pushing for acceptance in the mainstream of music theory.

Miranda, for example, is teaching classes on algorithmic composition at the Interdisciplinary Centre for Computer Music Research at the University of Plymouth [18]. David Cope continues work on `EMI`'s younger sister `Emily Howell`, a program that is capable of composing original pieces in a modern style rather than simply working in the realm of style replication [8]. Edwards is working on a program he calls `slippery chicken` to meld electronic musical thought into acoustic musical performance [13].

With these researchers (along with untold others, to be discovered) pursuing algorithmic composition, and with a strong history of algorithmic composers and researchers, algorithmic composition is far from dead. In fact, algorithmic composition seems to be in a sort of renaissance due to the increased computing power available to those who wish to wield it, and also due to the ever-increasing body of interested students looking for their next big project. Though the direction that algorithmic composition will take in the future is hard to tell, one thing is for certain: algorithmic composition is far from withering away into obscurity, and may in fact take a front seat position in the future of music. With increased reliance upon computers for recording and producing popular music today (one only has to notice the prevalence of Auto-Tuning on the radio to see this trend, or even to the increasingly popular dub-step genre of electronic dance music), it would not seem a far jump from production to composition, and with programs like `Emily Howell` and `slippery chicken` this might be a very real possibility in the near future.

5.3 Final comments

Algorithmic composition is an incredibly interesting topic to research, and anyone with an interest in music, computing, mathematics, and cognition would do extremely well to study it. A brief journey into the history of algorithmic composition will take the researcher through the realms of religion, mathematics, written language, emotion, predicate logic, mechanical computing, and even women's rights [28]. The algorithms discovered along the way are useful in several facets of everyday life, and provide inspiration in several other research opportunities, outside of algorithmic composition. Important people, places, facts, events, and works unfold like a book of tales untold, of pages unwritten, waiting for the correct interpretation to cement their place in the history of the world. And who provides that interpretation? The researcher.

To say that algorithmic composition is a simple idea is both truth and lie. Simple, yes, as what could be simpler than following concrete steps to a goal? However, complex, as minutiae and individual preference are always abundantly in the mix. You might easily spend years in pursuit of the ever elusive concept of "algorithmic composition," and all of a sudden, when least expecting it, algorithmic composition may present itself in the simple things like the arrangement of petals on a flower, or in the uniqueness of a human fingerprint.

The lesson here, to gather, is that algorithmic composition is a topic for serious study and inquiry, and it is also a topic to become intimately familiar with through experimentation and reiteration of previous research. But it is also a topic that will find you, whenever it suits its own fancy. Appreciate music for what it is, and algorithmic thought will uncover itself.

"Music is the pleasure the human soul experiences from counting without being aware that it is counting." —Gottfried Wilhelm von Leibniz (1646-1716), co-discoverer

of Calculus [23].

References

References

[1] Rita Aiello and John A. Sloboda, eds., *Musical Perceptions*, Oxford University Press, New York, 1994.

[2] Harold F. AtKisson, *Basic Counterpoint*, McGraw-Hill, New York, 1956.

[3] Gely P. Basharin, Amy N. Langville, and Valeriy A. Naumov, *The Life and Work of A. A. Markov*, Linear Algebra and its Applications 386, pages 3–26, 2004, `https://netfiles.uiuc.edu/meyn/www/spm_files/Markov-Work-and-life.pdf`.

[4] Alan Belkin, *General Principles of Harmony*, University of Montreal, 2008, `https://www.webdepot.umontreal.ca/Usagers/belkina/MonDepotPublic/bk.H/H.pdf`.

[5] Alan Belkin, *Principles of Counterpoint*, University of Montreal, 2008, `https://www.webdepot.umontreal.ca/Usagers/belkina/MonDepotPublic/bk.C/C.pdf`.

[6] David J. Benson, *Music: A Mathematical Offering*, Cambridge University Press, Cambridge, Massachusetts, 2007.

[7] John A. Biles, *GenJam: A Genetic Algorithm for Generating Jazz Solos*, International Computer Music Conference, 1994.

[8] Ryan Blitstein, *Triumph of the Cyborg Composer*, Miller-McCune, 2010, `http://www.miller-mccune.com/culture/triumph-of-the-cyborg-composer-8507/`.

[9] Noam Chomsky, *Three models for the description of language*, Information Theory, IEEE Transactions 2(3), pages 113–124, September 1956, `http://www.chomsky.info/articles/195609--.pdf`.

[10] David Cope, *Computers and Musical Style*, A-R Editions, Inc., Madison, Wisconsin, 1991.

[11] David Cope, *New Directions in Music, Seventh Edition*, Waveland Press, Inc., Prospect Heights, Illinois, 2001.

[12] Ben Coppin, *Artificial Intelligence Illuminated*, Jones and Bartlett Publishers, Inc., Sudbury, Massachusetts, 2004.

[13] Michael Edwards, *Algorithmic Composition: Computational Thinking in Music*, Communications of the ACM 54(7), pages 58–67, July 2011.

[14] Gordon Graham, *Philosophy of the Arts: an Introduction to Aesthetics*, Routledge, 2005.

[15] Leon Harkleroad, *The Math Behind the Music*, Cambridge University Press, Cambridge, Massachusetts, 2006.

[16] Lejaren A. Hiller and Leonard M. Isaacson, *Experimental Music: Composition With an Electronic Computer*, McGraw-Hill, New York, 1959.

[17] John E. Hopcroft and Jeffrey D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1979.

[18] ICCMR, *Computer Music Research*, Interdisciplinary Centre for Computer Music Research, 2011, `http://cmr.soc.plymouth.ac.uk/`.

[19] Bruce L. Jacob, *Composing with Genetic Algorithms*, International Computer Music Conference, 1995.

[20] G. R. Jones, T. I. Hempstock, K. A. Mulholland, and M. A. Stott, *Teach Yourself Acoustics*, The English Universities Press Ltd., London, 1967.

[21] David Koelle, *The Complete Guide to JFugue: Programming Music in Java*, First ed., Web, 2008.

[22] Daniel J. Levitin, *This Is Your Brain on Music*, Dutton, New York, 2006.

[23] G. Wilhelm Liebniz, *G. Wilhelm Liebniz quotes*, Thinkexist,com, `http://thinkexist.com/quotation/music_is_the_pleasure_the_human_mind_experiences/206750.html`.

[24] Stelios Manousakis, *Musical L-Systems*, Masters Thesis–Sonology, The Royal Conservatory, The Hague, 2006.

[25] Adam Marczyk, *Genetic Algorithms and Evolutionary Computing*, The Talk Origins Archive, 2004, `http://www.talkorigins.org/faqs/genalg/genalg.html`.

[26] Jaime Serquera and Eduardo R. Miranda, *Algorithmic Sound Composition Using Coupled Cellular Automata*, Interdisciplinary Centre for Computer Music Research, 2010.

[27] Melanie Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, 1998.

[28] Gerhard Nierhaus, *Algorithmic Composition: Paradigms of Automated Music Generation*, Springer, New York, 2009.

[29] Jacob M. Peck, *Artificial Life Music: An A-Life Approach to Generating Melodies Featuring Genetic Algorithms*, SUNY Oswego, 2011, `http://csc416.suspended-chord.info/pdf/alm-report-final.pdf`.

[30] Jacob M. Peck, *LCompose: An L-System Approach to Generating Music*, QUEST Symposium, 2011.

[31] Jacob M. Peck, *Networking Conway's Game of Life: Towards a dependency model for life and death*, SUNY Oswego, 2001.

[32] Przemyslaw Prusinkiewicz and Astrid Lindenmayer, *The Algorithmic Beauty of Plants*, Springer-Verlag, New York, 1990.

[33] Curtis Roads, *The Computer Music Tutorial*, MIT Press, Cambridge, Massachusetts, 1996.

[34] Heinrich Schenker, *Harmony*, Oswald Jones, trans., University of Chicago Press, Chicago, Illinois, 1954.

[35] Martin Supper, *A Few Remarks on Algorithmic Composition*, Computer Music Journal, 25(1), 2001.

[36] Andrea Valle, *Integrated Algorithmic Composition: Fluid systems for including notation in music composition cycle*, NIME, 2008.

[37] Eric W. Weisstein, *Cellular Automaton*, MathWorld—A Wolfram Web Resource, `http://mathworld.wolfram.com/CellularAutomaton.html`.

[38] Eric W. Weisstein, *Lindenmayer System*, MathWorld—A Wolfram Web Resource, `http://mathworld.wolfram.com/LindenmayerSystem.html`.

[39] Eric W. Weisstein, *Markov Chain*, MathWorld—A Wolfram Web Resource, `http://mathworld.wolfram.com/MarkovChain.html`.

[40] Eric W. Weisstein, *Rule 30*, MathWorld—A Wolfram Web Resource, `http://mathworld.wolfram.com/Rule30.html`.

[41] Eric W. Weisstein, *Rule 90*, MathWorld—A Wolfram Web Resource, `http://mathworld.wolfram.com/Rule90.html`.

[42] Allen Winold and John Rehm, *Introduction to Music Theory*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1979.

[43] Stephen Wolfram, *A New Kind of Science*, Wolfram Media, Champaign, Illinois, 2002.

[44] Stephen Wolfram, *Random sequence generation by cellular automata*, Advances in Applied Mathematics, 7(2), 1986.

Appendices

Companion website

Anyone perusing this thesis should be advised that there is a companion website to this text available at `http://cs.oswego.edu/~jpeck2/`. At this location may be found MIDI sequences of the examples presented in section 3, as well as user session demos, source code for the various projects, and a collection of links to useful references omitted from this text. This site will remain active for the forseeable future, and in the case of it being taken offline, this material shall be preserved in the form of a CD-ROM available at the SUNY Oswego Honors Office.