Metamathematics of Algorithmic Composition

Michael Gogins

michael.gogins@gmail.com
https://michaelgogins.tumblr.com

May 2025

This essay recounts my personal journey towards a deeper understanding of the mathematical foundations and computational complexity of algorithmic music composition. I do not spend much time on specific algorithms used by myself or other composers; rather, I focus on general issues such as fundamental limits and possibilities, by analogy with metalogic, metamathematics, and computability theory. I discuss implications from these foundations for the practice and future of algorithmic composition.

In 1983, when I was a returning undergraduate majoring in comparative religion at the University of Washington, I attended a lecture on fractals by Benoit Mandelbrot, discoverer of the set named after him (Mandelbrot 1982; Peitgen et al. 2004). Briefly, given the quadratic recurrence equation $z_{n+1} = z_n^2 + c$, the Mandelbrot set consists of all points c in the complex plane for which z, starting with $z = 0$, does not approach infinity as the equation is iterated. Then, given some particular point c in the Mandelbrot set, there is a Julia set consisting of all points z for which z_n does not approach infinity as the equation is iterated. Mandelbrot showed slides illustrating how any point in the Mandelbrot set can be used as the generating parameter of a Julia set, and how a plot of the neighborhood near a point in the Mandelbrot set closely resembles the plot of the corresponding Julia set (Lei 1990) (this has recently been proved (Kawahira and Kisaka 2018)). In short, the Mandelbrot set serves as a *parametric map* of all Julia sets. By now there is an extensive literature on the Mandelbrot set and Julia sets, and research continues.

Already at the time of Mandelbrot's lecture, I was developing an interest in computer music and algorithmic composition, in particular, algorithmic composition based on fractals. What occurred to me during the lecture is that if I zoomed into a plot of the Mandelbrot set, searching for interesting-looking regions, I could then plot the Julia set

for a point in that region, and I could then somehow translate that Julia set into a musical score (Gogins 1992a). I worked out two variations of this idea.

Mandelbrot Set/Julia Set The composer explores the actual Mandelbrot set. When a point and its neighborhood seem interesting, the composer selects that point, and the corresponding Julia set is first generated, and then translated to a musical score. The 2-dimensional plot of the Julia set is mapped more or less directly onto a 3-dimensional piano-roll type score, with the x axis representing time, the y axis representing pitch, and the color of a point in the plot representing choice of instrument.

Parametric map of Iterated Function Systems An iterated function system (IFS) is a Hutchinson operator, a set of contractive affine transformations, and iterating the operator upon any initial set takes that set to a fixed point that is a fractal (Barnsley and Demko 1985; Barnsley, 1993 [1988][a]). The Collage Theorem proves that an IFS can approximate any set as closely as desired (Barnsley and Sloan 1989; Barnsley, 1993 [1988][a]). It is easy to show that this method is *compositionally universal* (Gogins 1992a, In preparation), that is, capable of directly generating any finite score. It is also possible to generate a parametric map of any subset of IFS, but because the Hutchinson operators have many more than two parameters, generating a parametric map (an analogue of the plot of the Mandelbrot set in the complex plane) for dozens or hundreds of parameters requires the use of a Hilbert index (Patrick, Anderson, and Bechtel 1968). A Hilbert index maps an N-dimensional space, such as a plane, cube, or hypercube, to a 1- or 2-dimensional sequence of numbers. The index is constructed in such a way that neighboring points in the N-dimensional space have nearby indices in the 1- or 2-dimensional sequence. The key idea behind a Hilbert index is to recursively subdivide the N-dimensional space into smaller planes, cubes, or hypercubes, called cells. Each cell is assigned a unique index based on its position within the overall space. The subdivision process continues recursively until a desired level of detail is reached. To determine the Hilbert index of a specific point in the N-dimensional space, start with the largest cell that contains the point, level $j = 0$. Then, working in arithmetic to the base N, subdivide that cell into N sub-cells for level 1, and select the sub-cell that contains the point. If it is the *kth* sub-cell at level j, then add $((k + 1)/N)^{-j}$ to the index. Repeat this process recursively until the smallest cell containing the point is reached. The index of that cell is the Hilbert index for the point. Hilbert indices work because all metric spaces have the same cardinality; therefore, there is always a one-to-one mapping between points in any N-dimensional space and points on the line or, more usefully, the plane. In any event, the plot of the IFS is translated to a score in much the same way as for the plot of a Julia set.

I experimented with both of these methods, doing parametric composition by zooming in on interesting regions of the map, generating and rendering scores, exploring points near scores that seemed promising, and iterating this process. Unfortunately, the scores

I generated from Julia sets had too much of a sameness, and producing a parametric map of more than just a few points for IFS simply took way too much time. In other words, computing parametric maps for IFS quickly became *computationally intractable*.

As I continued to pursue algorithmic composition, I found that this question of intractability is not merely a practical problem, but has profound mathematical and philosophical roots, beginning with Pythagoras and continuing on through the hierarchy of complexity classes in theoretical computer science.

The Philosophical Context

Music has since Pythagoras (C. Huffman 2024; C. A. Huffman 2014) been understood by some as an intellectual paradigm and to reveal, through harmony that is both numerical and audible, the structure of reality. For this reason music was a central part of the *quadrivium*, the standard curriculum of liberal arts in Western higher education from late Antiquity through the Renaissance.

The project of understanding reality through number advanced from Pythagoras, through Leibniz' hope for a *characteristica universalis*, a symbolic language that could express all rational thought (Davis 2018), to the logicism of Russell (Tennant 2023), Hilbert (Zach 2023), and others, that sought to derive all mathematical truths from formal logic. In one of the major achievements of all philosophy, Kurt Gödel (Gödel 1986) demonstrated in his incompleteness theorems that logicism cannot be fully implemented, because there exist true statements of logic that cannot be proved. About five years later, Alan Turing proved his Halting Theorem: it is impossible for a computer to decide whether another, arbitrary computer program will halt. Working out the consequences of these theorems has taken researchers a long time, and that work continues.

A major result of this research been the elucidation of a provable hierarchy of complexity classes for problems that are solvable by computer (Arora and Barak 2009). Other important results are proofs there are completely abstract machines that are not incomplete; these are called super-Turing and depend, one way or another, on doing arithmetic with real numbers; almost all real numbers are not Turing computable, because their decimal representations never terminate.

The Complexity Classes

The complexity classes are based on the capabilities and limitations of Turing machines. A Turing machine is an abstract, idealized computer with infinite memory that can run for an infinite number of steps. A universal Turing machine is one that can emulate any other Turing machine. Anything that can be computed by a definite, step by step procedure can be computed by a universal Turing machine; this is the Church-Turing thesis (Copeland 2024). Our contemporary digital computers are universal Turing machines – or would be, if they had infinite memory and infinite time to run.

For our purposes, the important complexity classes, assuming that complexity is defined as run time on a universal Turing machine, are as follows.

Super-Turing (ST**)** Problems that have *mathematical* solutions, but the solutions are not *Turing computable.*

Recursively enumerable (RE**)** Also called semi-decidable. Problems where a solution can be verified by a Turing machine, but proving that a solution does not exist is not always possible. If a solution does exist, then a Turing machine can get as close as one likes to solving the problem.

Recursively computable (R**)** This is the same as Turing computable, or decidable. The problem can be decided in a finite number of steps; that is, the problem can be either solved, or proved unsolvable.

Non-deterministic polynomial time (NP**)** Recursively computable, but the the time to solve increases faster than any polynomial function of the size of the problem. However, a solution can be *verified* in polynomial time.

Deterministic polynomial time (P**)** Computable by a deterministic Turing machine; the time to solve is bounded by some polynomial function of the size of the problem.

The complexity classes, including ST, mirror central issues in philosophy and science. It is a primary open question of philosophy whether Nature herself is super-Turing. If so, then human thought, including musical composition, might as part of Nature also be super-Turing. If not, then human thought is at most RE and could be emulated as closely as one likes by a Turing machine.

Yet in either case, scientific theories are RE or less, because it must be possible to compare the predictions of the theory to observations of Nature that are finite in number and of only finite precision. Nobody can see how Nature or human thought being super-Turing could ever be proved, so it is widely held that any physical process can be effectively emulated by a Turing machine; this is the *physical* Church-Turing thesis (Aaronson 2005; Copeland 2024).

The closely related question whether NP is contained in P is one of the most important open questions in science. Most mathematicians and scientists believe, for a variety of overlapping reasons, that $\mathsf{P} \neq \mathsf{NP}$.

If $\mathsf{P} = \mathsf{NP}$ can be proved, it then becomes possible in principle to automatically solve all problems of a size that human beings can grasp. And this means that, given the problem of composing a certain piece of music to specification, a composer is not needed. Algorithmic composition alone can do the job. In terms of parametric composition, computing a finite subset of the parametric map would be in P.

But if $\mathsf{P} \neq \mathsf{NP}$ can be proved, then we will know that there are problems that computers simply cannot solve, but human beings *perhaps* can solve. And this means that, given the problem of composing a certain piece of music to specification, a composer is a good thing to have around. Not only that, but the use of algorithmic composition does not change the benefits of involving a composer. In terms of parametric composition, that is not because of the indexing, but because of the time required to evaluate a *piece* for each point in the parametric map. Recall that computing a piece as a Julia set may

be in NP; therefore, computing a parametric map of pieces also is in NP. However, looking up the set of parameters for a given point in the map is P. It may be only $O(log\ n)$ or, for multi-dimensional searches, $O(n^c), 0 < c < 1$.

If neither can be proved, then we will just never know.

Algorithmic Composition in Context

It is now possible to return to the question of parametric composition with more understanding. It turns out that parametric composition spans the entire hierarchy of complexity classes.

ST Braverman proved that some Julia sets, perhaps including some that might be encountered in parametric composition, are not even recursively enumerable (Braverman and Yampolsky 2006, 2009).

RE The Mandelbrot set, properly speaking, is not *recursively computable*, i.e. not Turing computable (Blum and Smale 1993). The plots that we make of the set are approximations. Note that incomputability in analysis is related to but not identical with incomputability in logic and theoretical computer science. Hertling showed that although the Mandelbrot set is not recursively computable, it is nevertheless *recursively enumerable* (Hertling 2005); given enough time, one can approximate the actual set as closely as one likes. Most likely all Julia sets that would be of musical interest can be approximated as closely as desired – are recursively enumerable.

NP It is also very likely that many of these Julia sets are not computable in polynomial time. If so, computing a parametric map that is big enough to be useful for composers might remain forever computationally intractable.

P But if P = NP then in principle it is possible to produce a useful parametric map in polynomial time. In that case, parametric composition might open up a vast new world for composers.

To restate the question, if P = NP, then a high-resolution parametric map of considerable subsets of compositions would, as it were, display symmetries and patterns, such as those found in the Mandelbrot set, that could assist in parametric composition. It would afford a God's-eye view of possible structure in music, and would partly overcome the irreducibility of algorithmic composition. In other words, P = NP could imply that algorithmic composition can be made *intelligible*.

But if P $\neq$ NP it would in practice not be possible to produce a high-resolution parametric map of any considerable subset of compositions. And this would fracture and obstruct the intelligibility of algorithmic composition.

Another significant aspect of parametric composition in particular, and of algorithmic composition in general, is *computational irreducibility*. Although any Julia set approximately resembles the neighborhood of the Mandelbrot set near its generating parameter, almost every Julia set is the chaotic attractor of its generating equation. Therefore, the

orbit of the Julia equation is *computationally irreducible* in the sense of Wolfram (Wolfram 1985), as proved by Zwirn (Zwirn 2015). The orbit of the equation cannot be determined by examining the equation, and it cannot except in trivial cases be determined even by mathematically analyzing the equation. In order to know the orbit, it is necessary to actually iterate the equation, that is, to run a program that computes the iterations. Even then, we can only obtain an approximation. What is true of Julia sets is true of IFS or, for that matter, of any algorithm that is not just rudimentary. In order to know what an algorithm actually does, one must actually run the algorithm.

Before further exploring the mathematical foundations of algorithmic composition, I will provide some additional background relating to different software systems for algorithmic composition, and to artificial intelligence, which can also be used to compose music.

Methods of Algorithmic Composition

To expand upon the concept, *algorithmic composition* is the use of computer software to write musical compositions that would be difficult or impossible to write without software. It does not, for example, include the use of notation software to write down music that one hears in one's head (as that can be done with paper and pencil), nor does it include the use of audio workstation software to record and even overdub music that one improvises (as that can be done with a tape recorder). In other words, algorithmic composition consists of all compositionl techniques that are *idiomatic* to the computer. Of course, there is not just one method of algorithmic composition (Fernández and Vico 2013; Ariza 2023). A recent summary can be found in (McLean and Dean 2018). However, there is an obvious overlap with a more generic notion of *process music* or *generative music*, including Mozart's musical dice game (humdrum.org 2023), the minimalism of Steve Reich (Reich 2023; Schwarz 1980) and Philip Glass (Potter 2002; Glass 2015), and the generative work of Brian Eno (Eno 2023). The commonality between algorithmic composition and process music is precisely the simplicity and clarity of the means versus the complexity and unpredictability of the results; in other words, yet again, irreducibility.

Here I should clarify this idea of irreducibility. It is not a binary choice, it is a spectrum. The minimum of irreducibility occurs when the composer simply writes down what he or she hears in his or her imagination. The maximum occurs when the composition is generated in an entirely random way, so that there is absolutely no way for the composer to predict, better than chance, any particular note or sequence of notes; but even then, there is a degree of musical intelligibility in that the texture of one random variable (e.g. white noise) can easily be distinguished from the texture generated by another random variable (e.g. brown noise). In the middle of the spectrum is an area where the composer does have some degree of insight into the kind of music that will be generated, even though the details cannot be predicted. And this is the most interesting and most useful degree of computational irreducibility.

Hiller and Isaacson's *Illiac Suite* (Lejaren A. Hiller and Isaacson 1957) is the first

piece of what can unambiguously be called computer music, and it is an algorithmic composition assembled using a toolkit of stochastic note generators and music transformation functions, as detailed in their book *Experimental Music* (Hiller and Isaacson 1959). This can be called the *toolkit approach* to algorithmic composition. The composer programs a chain of generators, including random variables, and transformations, including serial transformations, to produce a chunk of music. The chunks can then be edited by hand. Multiple chunks can be assembled into a composition by hand. The toolkit approach lives on in contemporary software systems such as Open Music (C. A. Agon, Gerard Assayag, and Bresson 2008), Common Music (Taube 2023, 2021), and many others. This is to date the most successful and widely used method of algorithmic composition.

The more recent method of algorithmic composition known as *live coding* can be considered a variant of the toolkit approach. A live coding system for music consists of a toolkit of routines that are assembled into a music-generating graph during a live performance by interpreting real-time commands in a domain-specific language. Live coding systems have tools and commands for both high-level representations of music (notes, loops, chords, scales, musical transformations, etc.) and sound synthesis (oscillators, envelope generators, filters, etc.). An overview of the field can to some extent be gleaned from the TOPLAP web site (toplap.org 2023) and the *Oxford Handbook of Algorithmic Composition* (McLean and Dean 2018). I have some experience with Tidal-Cycles (computer platform, Haskell implementation) (McLean et al. 2023) and Strudel (a JavaScript version of TidalCycles, Web browser platform, JavaScript implementation) (Roos et al. 2023).

Some composers, such as myself, prefer to use an algorithm, such as a Lindenmayer system (Prusinkiewicz and Lindenmayer, 1996 [1991]; Prusinkiewicz 1986; Gogins 1992b) or iterated function system (IFS) (Barnsley, 1993 [1988][b]; Gogins 1991) that will generate an entire piece based on fractals or other mathematical methods, without further editing or assembling. This can be called the *fractal approach* to algorithmic composition.

Recently it has become possible to compose music using large language models (LLMs) such as ChaptGPT. This can be called the *machine learning* approach to computer music. (I prefer the term *machine learning* to *artificial intelligence* because the software is not intelligent, but it is trainable and so in some sense it is learning.) I discuss only LLMs as they are currently the most influential method of machine learning. Briefly, the approach is based on emulating biology, specifically, on simulating at a high level of abstraction the behavior of nerve cells. A neural network is built up from layers of simulated neurons that connect with each other; the connections have tunable weights or parameters that control the output behavior of neurons in one layer given inputs from connected neurons in other layers. First the data, a large corpus of texts, is syntactically broken up into tokens. "Attention heads" are trained (in parallel) to learn weights indicating how tokens within the entire data set are related syntactically and semantically. Each head is trained to reflect some different aspect of the data. The trained heads are then used to transform the input (the prompt) *plus* current output (hence the term "*self*-attention") to a new input from which the next token of output is predicted (Vaswani et al. 2017). This attention mechanism, and other heuristic mechanisms, have

been found to greatly increase the power of the neural networks. In particular, the attention mechanism makes it possible to train the network on a very large body of data more efficiently and without much human intervention. For more detail, see (Zhang et al. 2023) and OpenAI's paper on their current LLM architecture (OpenAI 2023a). For working examples of how LLMs can be used to compose music, see Jukebox (OpenAI 2023b), Gonsalves (Gonsalves 2021), and Ocampo et al. (Ocampo et al. 2023).

Although it is early days for machine learning, contemporary experience has led to a number of reviews and critical studies of the capabilities and limitations of machine learning. From the skeptical side, see (Dale 2021). For an amusing and instructive series of dialogues between all sides, see (Aaronson 2023). This experience makes it possible to identify a few more important things about the mathematical foundations of algorithmic composition:

Computational opacity All agree that LLMs can generate amazing, even spooky, results without anyone understanding much about what is going on in the neural network. We have a perfect understanding of each component in the LLM, because these components are actually quite simple, but we have no idea at all how something like ChatGPT (a particular brand of LLM) can conduct a fact-filled conversation with one in perfect English. The details are scattered through billions of neural network weights in the LLM. Computational opacity is a form of computational irreducibility, but it goes far beyond irreducibility because, with computational opacity, we cannot obtain even a partial understanding of the actual computations performed by the software. We have taken one irreducible program (the untrained LLM) and used it to build another irreducible program (the trained LLM)! Even though an untrained LLM is computationally irreducible, we still have a perfect understanding of how it actually works; but it seems very likely that we will not, at least in practice, ever obtain even a partial understanding of how the *trained* LLM actually works.

Hallucination Refers to the tendency of LLMs to generate factually incorrect responses to prompts. It is a reminder that the software has no sense of reality and no means of comparing what it generates with the real world. I suspect that hallucinations arise from the human mistakes, conflicting goals, and outright lies represented in the training data. Ways of dealing with hallucinations are being investigated; for one approach, see (Christiano et al. 2017).

Unoriginality LLMs generate responses to prompts based on high-dimensional correlations that the LLMs have automatically discovered in the training data – data that *we* have provided. This is a self-referential situation. When we converse with ChatGPT we are looking at ourselves in a mirror; and in fact, in a fun-house mirror.

Artistic Results and Procedures

To date, not many pieces of algorithmically composed music have become popular or influential, even among composers and aficionados of art music and experimental music. A few of the pieces that have been influential include Iannis Xenakis' *La Légende D'Eer* (Xenakis 2005) and *Gendy 3* (Xenakis 1995), Charles Dodge's *Viola Elegy* (Dodge, April 1994), and some of Brian Eno's works (Eno 2023; Eno and Chilvers 2023). I have my own idiosyncratic list of different algorithmic composition systems, with my own choice of representative pieces (Gogins 2019).

The actual procedures followed by composers for algorithmic composition vary by genre, by composer, and by the software used. It is difficult to get a handle on the actual practices of any composers, let alone algorithmic composers. Composing can be a communal effort, as in much contemporary popular music, but art music is usually rather private, and algorithmically composed music even more so. However, IRCAM has published a series of books with chapters by composers on how they have used OpenMusic (C. Agon, Gérard Assayag, and Bresson 2006-2016, 2006a, 2006b, 2016). These are very useful. Profiles of composers in *Computer Music Journal* also can be useful. Here I will explain the general procedure that I myself follow.

I start with some kind of mathematical system that can be used to generate a set of musical notes, a score. The system needs to generate complex structure that can be changed by varying a relatively small number of numerical parameters or commands. It's often useful to select a recursive algorithm that, as the number of iterations approaches infinity, approaches a fixed point that is a fractal.

Such generative algorithms generally reflect processes in Nature that produce fractal-like forms, such as the patterns on seashells or the branching of plants. I have used chaotic dynamical systems, Lindenmayer systems, iterated function systems, and other systems.

Generally speaking, how to set the parameters in order to obtain a desired result is more or less opaque. This is well-known as the *inverse problem* (Graham and Demers 2021; Tu et al. 2023). But actually this is another form, once again, of computational irreducibility, meaning in this case that it is not intuitive how to infer the structure of an algorithm even after closely inspecting its results.

On the one hand this is a fault of the method; but on the other hand, and even more so, it is a virtue. In this way, and only in this way, can we generate scores that we would not otherwise be able to imagine. This, of course, is another fundamental motivation for pursuing algorithmic composition. And it's the most important motivation. *This kind of algorithmic composition actually amplifies our musical imagination.*

Now the question arises, how can such opaque methods be used to compose good music? It is difficult but by no means impossible, and here is the usual way I do it.

The parameters generally have a global effect on the final structure, that is, on the generated score. For example, an iterated function system consists of a number of affine transformations that are repeatedly applied to a set. Changing just one element of one transformation matrix can easily change every note in the generated score.

So, I pick one parameter and change its value, then listen to the result. I change the

value again, and listen to the second result. Then, I choose the value that I prefer. I make more changes and listen again. Eventually I will find a value for that parameter that is more or less optimal – a sweet spot in the parameter space.

Then I pick another parameter and change its value in the same way, until I have a second sweet spot. During this process, the effect of the first parameter will probably change, so I go back to the first parameter and search again for the "sweet spot."

This can be repeated for any number of parameters, but it is a tedious process and does not make sense for more than a few parameters.

This procedure amounts to a sort of binary search through a set of possible parameter values so vast – indeed infinite – that a linear search is simply out of the question. But a binary search is far more efficient than a linear search. Furthermore, finding two or three "sweet spots" in a small set of controlling parameters – each of which has global effects on the entire score – can produce a surprisingly large improvement in the musicality of the result.

I see here an analogy with the way in which LLMs work. There are repeated searches in a parameter space equipped with a fitness function (as with the attention mechanism) at increasing levels of refinement (as with gradient descent).

I have known a number of composers, some quite well. Few algorithmic composers simply "hear music in their heads" and write software to render it, although that certainly happens. Most fool around producing various experimental chunks of music, refine them more or less as I have described, and assemble some of them into a finished composition.

Before I proceed to look at this kind of production from a mathematical point of view, I will summarize what I have learned about the mathematical foundations:

Incomputability The set of possible musical compositions (assuming that some pieces either last an infinitely long time, or that between any two pieces is a continuous path consisting of variations between the pieces) is *recursively incomputable.*

Universality In spite of the incomputability of compositions, they are *recursively enumerable*, so it is possible to approximate any possible composition as closely as one likes.

Irreducibility Compositional algorithms that have a strong analogy to Julia sets are *computationally irreducible.*

Opacity Compositional algorithms based on machine learning are not only computationally irreducible, but also computationally opaque in that we have essentially no insight into the meaning of the steps followed by the LLM.

Mappability Compositional algorithms are nevertheless always mappable. This ultimately is because the compositions can be ordered in some way, either as sets of scores, or as sets of generating parameters, and individually evaluted.

Intractability Producing a useful parametric map of some subset of compositions is very compute-intensive.

Hallucination LLMs that are supposed to provide true or useful outputs sometimes just make stuff up. But this means that material generated by an LLM in response to a prompt cannot be trusted to be true or useful. It is necessary for a person, indeed an expert, to evaluate the material. It is by no means clear at this time whether an expert equipped with a LLM is more productive for creative work than that same expert without the LLM.

Unoriginality LLMs work by discovering high-dimensional correlations in large bodies of training data. This means that LLMs can select, summarize, and vary — but they cannot generate an output that is not correlated with the training data. In other words, there is a limit to their originality. However, it is by no means clear at this time whether that limit is well below, or well above, the creativity of experts in the field from which the training data was drawn.

I will now put forward some conjectures based on these foundations.

Limitations

At this time and for the foreseeable future, no form of artificial intelligence is conscious or has its own goals. Therefore, for the foreseeable future, human composers must and will play a irreplaceable role in algorithmic composition. This involves selecting a subset of possible compositions to study, evaluating the musical quality of each composition in the subset, and varying the parameters or prompts that generate the pieces. This follows from hallucination and unoriginality.

Incomputability, irreducibility, and opacity set objective limits on how much understanding composers can gain into the working of their algorithms and of the music generated by them. This is both a limitation and an advantage. In practice, it is not possible to determine in advance just where those limits lie.

Sophisticated forms of algorithmic composition are compute-intensive, and can be computationally intractable.

Prospects

Computer power will continue to increase. This will most likely make algorithmic composition both more productive and more important.

There is a similarity between a composer's experience with a toolkit of algorithms, the transformation of prompts into responses by an LLM, and exploring the parametric map of a fractal compositional algorithm. In all cases, starting with an initial trial, a final composition is approached by a descending, zigzag search through a space representing musical possibilities of differing value, until the search comes to rest in some local optimum.

Every method that establishes a more musically compact and/or intelligible ordering of a space of musical possibilities will make algorithmic composition more productive. As an example, excluding non-musical elements from the space of possibilities can be very helpful, because sounds that human beings consider to be musical occupy a vanishingly

small part of the parameter space of universal compositional algorithms. However, it is possible to literally change the mathematical basis of the parameter space to represent only musically pertinent features. For example, rather than representing scores as notes on piano rolls, e.g. planes or cubes, one can represent scores as more or less fleeting chord progressions in chord spaces (Gogins 2006, In preparation).

Every method that speeds up the search process will make algorithmic composition more productive. In particular, the growth of live coding demonstrates that the toolkit approach to algorithmic composition has a future. The underlying reason is that live coding supports faster searching, due to concise commands and immediately audible feedback. Spending time doing live coding also increases the composer's insight into the tools.

In Sum

The main result here is that the major approaches to algorithmic composition — trial and error with a toolkit of algorithms, live coding, exploration of fractals, and machine learning — share this fundamental business of zigzagging down a slope on a landscape of evaluations to rest in a local optimum. This result is proved by the simple fact that the generated music and/or the parameters used to generate it can be ordered. The dimensionality of the musical space is secondary, as it can be reduced to one or two dimensions by means of a Hilbert index. Note that searching for solutions or optimizations in many domains is known to be computationally expensive.

Future developments in artificial intelligence may have a significant impact on algorithmic composition. For example, machine learning has been applied to solving the inverse problem for discovering the parameters of fractal algorithms (Tu et al. 2023). It might then be possible to represent an existing score, or scores, as fractal parameters and then work with these parameters to vary or interpolate between such pieces. This does not overcome computational irreducibility, as it substitutes the opacity of machine learning for the irreducibility of the inverse problem, yet it still might be very useful.

Algorithmic compositions based on current LLMs are easier to produce, but will not usually be musically original; while algorithmic compositions based on toolkits, live coding, or fractals can be musically original, but are inherently more difficult to produce.

As for algorithmic compositions based on fractals, sometimes an analytical understanding of the mathematics can be used as a guide to composition, but this tends in my experience to be of limited use. More often, the only way to penetrate the fog of incomputability, irreducibility, opacity, and intractability is to explore the geometrical order in a subset of compositions. This can be done either by trial and error, or by literally plotting a map of the subset of compositions. One might say that with trial and error one plots a sparse map of fully defined features in a territory, and with a parametric map one explores a densely mapped territory with partially defined features.

Progress in algorithmic composition seems likely to depend on speeding up the composer's workflow, whether in parametric composition, in live coding, in algorithmic composition toolkits, or in machine learning; and, even more so in the long run, on defining more musically compact and intelligible spaces of musical possibility.

To state this more clearly, the parameters, prompts, or other inputs to the compositional algorithms can be refined such that they are musically intelligible. The processes are not just mathematical, they have a musical meaning. This in my view is the task of future algorithmic composers: identify and refine such processes.

There are several features of the Mandelbrot set/Julia set duality that are important for a deeper understanding of algorithmic composition not just for these fractals, but for all methods of algorithmic composition.

Generative Music Theory

The toolkit of algorithms for composing music includes transformations and generators from many sources:

Traditional Music Theory Canonical forms, scales, transpositions, and rescalings of pitch and time.

"Atonal Theory" or "Set Theory"' Starting with transpose, invert, and reverse, continuing on to other group operations on tone rows or other pitch collections.

Neo-Riemannian Theory Mathematizes voice-leading and discovers groups consisting of notes, scales, and chords, together with their symmetries, and operations upon them.

Random Variables Borrowed from mathematics and used to generate series of musical properties such as pitches.

Dynamical Systems Borrowed from mathematics and used to generate series of musical properties such as pitches.

Fractals Borrowed from mathematics and used to generate some or all aspects of musical scores.

As long as $P \neq NP$, for the reasons listed above, compositional algorithms will not be broadly mappable, and will thus remain somewhat opaque and unintelligible. However, there is a critical difference between applying, *e.g.*, a series of transformations from the General Contextual Group of Fiore and Satyendra, even if the transformations are selected at random, and simply applying a random variable to scramble or transpose a chord. Any application of the GCG is going to generate a chord progression that sounds more or less musically well-formed, because the group instantiates an analysis of some aspects of neo-Riemannian theory. Simply applying a random variable will sound, well, random, with perhaps some musical color from the nature of the random distribution.

In other words, while both procedures are computationally irreducible, the GCG procedure will generate far more musically intelligible results.

Look for results in mathematical music theory that can be turned into algorithmic generators.

Parametric map of rule 110

The fixed point of consciousness is not finite

Discrete dynamical systems can perform universal computation (e.g. the rule 110 one-dimensional cellular automaton, or the two-dimensional Game of Life).

Continuous dynamical systems can emulate discrete dynamical systems, so also can perform universal computation. But can a continuous dynamical system perform super-Turing computation?

References

Aaronson, Scott. 2005. Np-complete problems and physical reality. *ACM SIGACT News* 36 (1): 30–52.

———. 2023. *Shtetl-optimized: the blog of scott aaronson.* Available online at: `https://scottaaronson.blog/`. Accessed 10 May 2023.

Agon, Carlos, Gérard Assayag, and Jean Bresson. 2006a. *The om composer's book 1.* Vol. 2006. Editions Delatour France/Ircam-Centre Pompidou.

———. 2006b. The om composer's book 2. *Editions Delatour/IRCAM Centre Pompidou* 2008.

———. 2016. The om composer's book 3. *Editions Delatour/IRCAM Centre Pompidou* 2016.

———. 2006-2016. *The om composers book.* Available online at: `http://repmus.ircam.fr/openmusic/ombook`. Accessed 11 May 2023.

Agon, Carlos Augusto, Gerard Assayag, and Jean Bresson. 2008. *Openmusic.* `Http://recherche.ircam.fr/equipes/repmus/OpenMusic/`.

Ariza, Christopher. 2023. *Algorithmic.net.* Available online at: `http://algorithmic.net/`. Accessed 13 May 2023.

Arora, Sanjeev, and Boaz Barak. 2009. *Computational complexity: a modern approach.* Cambridge University Press. ISBN: 9780521424264. `https://theory.cs.princeton.edu/complexity/book.pdf`.

Barnsley, Michael F, and Stephen Demko. 1985. Iterated function systems and the global construction of fractals. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 399 (1817): 243–275.

Barnsley, Michael F, and Alan D Sloan. 1989. Fractal image compression. In *Proc. scientific data compression workshop,* 351–365.

Barnsley, Michael F. 1993 [1988](a). *Fractals everywhere.* 2nd. Boston: Academic Press Professional.

———. 1993 [1988](b). *Fractals everywhere.* 2nd. Boston: Academic Press Professional.

Blum, Lenore, and Steve Smale. 1993. The gödel incompleteness theorem and decidability over a ring. In *From topology to computation: proceedings of the smalefest,* 321–339. Springer.

Braverman, Mark, and Michael Yampolsky. 2006. Non-computable julia sets. *Journal of the American Mathematical Society* 19 (3): 551–578.

———. 2009. *Computability of julia sets.* Vol. 23. Available online at: `https://arxiv.org/pdf/math/0610340.pdf`. Accessed 11 May 2023). Springer.

Christiano, Paul F, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. *Advances in neural information processing systems* 30.

Copeland, B. Jack. 2024. The Church-Turing Thesis. In *The Stanford encyclopedia of philosophy,* Winter 2024, edited by Edward N. Zalta and Uri Nodelman. Metaphysics Research Lab, Stanford University.

Dale, Robert. 2021. Gpt-3: what's it good for? *Natural Language Engineering* 27 (1): 113–118.

Davis, Martin. 2018. *The universal computer: the road from leibniz to turing.* 3rd. Boca Raton, FL: CRC Press. ISBN: 978-1138502086.

Dodge, Charles. April 1994. Any Resemblance is Purely Coincidental. Chap. Viola Elegy. NA043. New Albion Records.

Eno, Brian. 2023. *Generative music.* Available online at: `https://inmotionmagazine.com/eno1.html`. Accessed 10 May 2023.

Eno, Brian, and Peter Chilvers. 2023. *Bloom: 10 worlds by brian eno and peter chilvers. world 1: origin.* Available online at: `https://music.youtube.com/watch?v=rGAUQmfV6w4`. Accessed 10 May 2023.

Fernández, Jose D, and Francisco Vico. 2013. Ai methods in algorithmic composition: a comprehensive survey. *Journal of Artificial Intelligence Research* 48:513–582.

Glass, Philip. 2015. *Words without music: a memoir.* WW Norton & Company.

Gödel, Kurt. 1986. On formally undecidable propositions of *Principia Mathematica* and related systems i. In *Kurt gödel: collected works, volume i: publications 1929–1936,* edited by Solomon Feferman, John W. Dawson Jr., Stephen C. Kleene, Gregory H. Moore, Robert M. Solovay, and Jean van Heijenoort, 144–195. Originally published in German in 1931. Oxford University Press. ISBN: 9780195039641.

Gogins, Michael. 1991. Iterated Function Systems Music. *Computer Music Journal* 15 (1): 34–42.

———. 1992a. ...how i became obsessed with finding a mandelbrot set for sounds. *News of Music* 13:129–139.

Gogins, Michael. 1992b. Fractal Music with String Rewriting Grammars. *News of Music* 13:146–170.

————. 2006. Score generation in voice-leading and chord spaces. In *Proceedings of the 2006 international computer music conference,* edited by Georg Essl and Ichiro Fujinaga. San Francisco, California: International Computer Music Association.

————. 2019. *More rant-like musings on algorithmic composition software.* Available online at: `https://michaelgogins.tumblr.com/post/188345210028/algorithmiccompositionsystems`. Accessed 9 May 2023.

————. In preparation. Parametric composition of score graphs.

Gonsalves, Robert A. 2021. *Ai-tunes: creating new songs with artificial intelligence.* Available online at: `https://towardsdatascience.com/ai-tunes-creating-new-songs-with-artificial-intelligence-4fb383218146`. Accessed 9 May 2023.

Graham, Liam, and Matthew Demers. 2021. Applying neural networks to a fractal inverse problem. In *Recent developments in mathematical, statistical and computational sciences: the v ammcs international conference, waterloo, canada, august 18–23, 2019,* 157–165. Springer.

Hamilton, Chris H, and Andrew Rau-Chaplin. 2007. Compact hilbert indices for multidimensional data. In *First international conference on complex, intelligent and software intensive systems (cisis'07),* 139–146. IEEE.

Hertling, Peter. 2005. Is the mandelbrot set computable? *Mathematical Logic Quarterly* 51 (1): 5–18. `https://doi.org/10.1002/malq.200310124`.

Hiller, Lejaren, and L.M. Isaacson, eds. 1959. *Experimental music: composition with an electronic computer.* New York, New York: McGraw–Hill.

Huffman, Carl. 2024. Pythagoras. In *The Stanford encyclopedia of philosophy,* Spring 2024, edited by Edward N. Zalta and Uri Nodelman. Metaphysics Research Lab, Stanford University.

Huffman, Carl A. 2014. *A history of pythagoreanism.* Cambridge University Press.

humdrum.org. 2023. *Musikalisches würfelspiel.* Available online at: `https://dice.humdrum.org/`. Accessed 11 May 2023.

Kawahira, Tomoki, and Masashi Kisaka. 2018. Julia sets appear quasiconformally in the mandelbrot set. *arXiv preprint arXiv:1804.00176.*

Lei, TAN. 1990. Similarity between the mandelbrot set and julia sets. *Commun. Math. Phys* 134:587–617.

Lejaren A. Hiller, Jr., and Leonard M. Isaacson. 1957. *Illiac suite for string quartet.* New York: New Music Editions.

Mandelbrot, Benoit B. 1982. *The fractal geometry of nature.* W. H. Freeman, August. ISBN: 0716711869. http://www.amazon.ca/exec/obidos/redirect?tag=citeuli ke09-20%5C&path=ASIN/071%206711869.

McLean, A., and R.T. Dean. 2018. *The oxford handbook of algorithmic music.* Oxford Handbooks. Oxford University Press. ISBN: 9780190227005. https://books.googl e.com/books?id=7XBGDwAAQBAJ.

McLean, Alex, et al. 2023. *Tidal cycles.* Available online at: https://tidalcycles. org/. Accessed 14 May 2023.

Ocampo, Rodolfo, Josh Andres, Adrian Schmidt, Caroline Pegram, Justin Shave, Charlton Hill, Brendan Wright, and Oliver Bown. 2023. Using gpt-3 to achieve semantically relevant data sonificiation for an art installation. In *Artificial intelligence in music, sound, art and design: 12th international conference, evomusart 2023, held as part of evostar 2023, brno, czech republic, april 12–14, 2023, proceedings,* 212–227. Springer.

OpenAI. 2023a. *Gpt-4 technical report.* arXiv: 2303.08774 [cs.CL].

———. 2023b. *Jukebox.* Available online at: https://openai.com/research/jukebox/. Accessed 10 May 2023.

Patrick, Edward A, Douglas R Anderson, and Friend K Bechtel. 1968. Mapping multi-dimensional space to one dimension for computer output display. In *Proceedings of the 1968 23rd acm national conference,* 511–515.

Peitgen, Heinz-Otto, Hartmut Jürgens, Dietmar Saupe, Heinz-Otto Peitgen, Hartmut Jürgens, and Dietmar Saupe. 2004. The mandelbrot set: ordering the julia sets. *Chaos and Fractals: New Frontiers of Science,* 783–837.

Potter, Keith. 2002. *Four musical minimalists: la monte young, terry riley, steve reich, philip glass.* Vol. 11. Cambridge University Press.

Prusinkiewicz, P. 1986. Score generation with L-systems. In *Proceedings of the 1986 international computer music conference,* 455–457.

Prusinkiewicz, Przemyslaw, and Artistid Lindenmayer. 1996 [1991]. *The algorithmic beauty of plants.* Available online at http://algorithmicbotany.org/papers. New York: Spring-Verlag.

Reich, Steve. 2023. *Music as a gradual process.* Available online at: https://static1.s quarespace.com/static/50e79ec7e4b07dba60068e4d/t/515707b0e4b0ec1768d 61b17/1364658096256/Reich.pdf. Accessed 11 May 2023.

Roos, Felix, et al. 2023. *Strudel.* Available online at: https://strudel.tidalcycles. org/. Accessed 14 May 2023.

Schwarz, K. Robert. 1980. Steve reich: music as a gradual process: part i. *Perspectives of New Music* 19 (1/2): 373–392. ISSN: 00316016, accessed May 11, 2023. `http://www.jstor.org/stable/832600`.

Taube, Rick. 2021. *Musx.* Available online at: `https://github.com/musx-admin/musx`. Accessed 9 May 2023.

———. 2023. *Common Music.* `Https://commonmusic.sourceforge.net/`.

Tennant, Neil. 2023. Logicism and Neologicism. In *The Stanford encyclopedia of philosophy,* Winter 2023, edited by Edward N. Zalta and Uri Nodelman. Metaphysics Research Lab, Stanford University.

toplap.org. 2023. *Toplap.* Available online at: `https://toplap.org/about/`. Accessed 14 May 2023.

Tu, Cheng-Hao, Hong-You Chen, David Carlyn, and Wei-Lun Chao. 2023. Learning fractals by gradient descent. *arXiv preprint arXiv:2303.12722.*

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30.

Wolfram, Stephen. 1985. Undecidability and intractability in theoretical physics. *Physical Review Letters* 54 (8): 735.

Xenakis, Iannis. 1995. Xenakis: Ais, Gendy3, Taurhiphanie, Thallein / Nee, Steiger. Chap. Gendy3. 45086. Neuma.

———. 2005. *La légende d'eer.* Edited by Gerard Pape and Iannis Xenakis. Xenakis, Iannis, 1922-2001. Electronic works ; Program notes in English, French, and German ([21] p. : ill.) in container. New York, NY: Mode.

Zach, Richard. 2023. Hilbert's Program. In *The Stanford encyclopedia of philosophy,* Spring 2023, edited by Edward N. Zalta and Uri Nodelman. Metaphysics Research Lab, Stanford University.

Zhang, Chaoning, Chenshuang Zhang, Sheng Zheng, Yu Qiao, Chenghao Li, Mengchun Zhang, Sumit Kumar Dam, Chu Myaet Thwal, Ye Lin Tun, Le Luang Huy, et al. 2023. A complete survey on generative ai (aigc): is chatgpt from gpt-4 to gpt-5 all you need? *arXiv preprint arXiv:2303.11717.*

Zwirn, Herve. 2015. Computational irreducibility and computational analogy. *HAL* 2015.