

Metamathematics of Algorithmic Composition

Michael Gogins^a

^aIrreducible Productions, 1576 Crescent Valley Road, Bovina Center, New York 13740, United States

ARTICLE HISTORY

Compiled May 8, 2025

ABSTRACT

I recount my journey towards a deeper understanding of the philosophical context, mathematical foundations, and computational complexity of algorithmic music composition. I discuss different types of algorithms, but my primary focus is on the fundamental limits and possibilities of algorithmic composition, by analogy with metalogic, metamathematics, and computability theory. I present implications for the practice and future of algorithmic composition.

KEYWORDS

music theory; algorithmic composition; computability theory; computational complexity

1. Introduction

In 1983, as a returning undergraduate in comparative religion at the University of Washington, I attended a lecture on fractals by Benoit Mandelbrot, discoverer of the eponymous set (Mandelbrot, 1982; Peitgen et al., 2004). Briefly, given the quadratic recurrence $z_{n+1} = z_n^2 + c$, the Mandelbrot set consists of all points c in the complex plane for which z , starting at 0, does not approach infinity as the equation is iterated. Then, for each point c in the Mandelbrot set there is a connected Julia set, consisting of all points z for which z_n does not approach infinity as the equation is iterated. Mandelbrot's slides showed how a plot of the neighborhood near some c in the Mandelbrot set closely resembles the plot of the corresponding Julia set (Lei (1990); recently proved by Kawahira and Kisaka (2018)). In short, the Mandelbrot set is a *parametric map* of all connected Julia sets. There is now an extensive and growing literature on the Mandelbrot set and Julia sets.

I was already interested in computer music, especially algorithmic composition based on fractals. During the lecture it occurred to me that if I zoomed into a plot of the Mandelbrot set, searching for interesting-looking regions, I could plot the Julia set for a point in that region, and then somehow translate that Julia set into a musical score (Gogins, 1992b). I have worked out several variations of this idea.

1.1. Mandelbrot Set/Julia Set

The composer explores the Mandelbrot set. When a region seems interesting, the composer selects a point in it, 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 onto a 3-dimensional piano-roll score: the x axis is time, the y axis is pitch, and the z axis is the choice of instrument, also indicated by color.

1.2. Parametric Map of Iterated Function Systems

An iterated function system (IFS) is a Hutchinson operator, a set of contractive affine transformations, each of which is applied to transform an initial set; the union of these transformations becomes the initial set for the next iteration. Iterating the operator upon *any* initial set takes that set to the *same* fixed point, which is a fractal (Barnsley, 1993 [1988]; Barnsley & Demko, 1985). The Collage Theorem proves that an IFS can approximate any set as closely as desired (Barnsley, 1993 [1988]; Barnsley & Sloan, 1989). Therefore this method is *compositionally universal* (Gogins, 1992b, In preparation): it can generate, as closely as one likes, any finite score.

It is also possible to generate a parametric map of any subset of IFSs, but as Hutchinson operators have more than two parameters, generating a parametric map (an analogue of the plot of the Mandelbrot set) for dozens or hundreds of parameters requires the use of a Hilbert index, which maps points in an N -dimensional space, such as a plane, cube, or hypercube, to a 1- or 2-dimensional sequence of numbers (Patrick, Anderson, & Bechtel, 1968).

There is one dimension for each IFS parameter. The index is constructed so that neighboring points in the N -dimensional space usually have neighboring entries in the 1- or 2-dimensional index. The key idea 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. Subdivision 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 containing 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 containing the point. If it is the k th 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 in \mathbb{R}^n have the same cardinality; hence 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. 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 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. However, I found that scores generated directly from Julia sets had too much of a sameness. And producing a parametric map of more than just a few points for IFSs simply took way too much time. In other words, computing parametric maps for IFSs is *computationally intractable*.

As I pursued algorithmic composition, I found that 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. I found only a few discussions of these issues of complexity and intractability in the computer music literature.

2. 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 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. Five years later, Alan Turing proved the Halting Theorem: it is impossible for any computer program to decide whether another, arbitrary computer program will halt. Researchers are still exploring the consequences of these theorems. Some major results:

- The elucidation of a provable hierarchy of complexity classes for problems that are solvable (or not) by computer (Arora & Barak, 2009).
- Proofs that there are abstract (so far completely unphysical) computers 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 uncomputable, because their decimal representations never terminate) (Ord, 2006).
- Proofs that the relationship between energy and information imposes hard *physical* limits on the amount of computation that can be done in the physical universe (Aaronson, 2005; Piccinini & Maley, 2021).

2.1. Complexity Classes

The complexity classes are based on the capabilities and limitations of Turing machines. A TM is an abstract, idealized computer with infinite memory that can run for an infinite number of steps. It may halt after a finite number of steps and produce a result, or never halt. A *universal* TM can emulate any other TM. The widely accepted Church-Turing thesis holds that *anything* computable by a definite, step by step procedure is computable by a universal TM (Copeland, 2024). Contemporary digital computers are universal TMs — or would be, if they had infinite memory and infinite time to run.

For our purposes, the important complexity classes are:

- ST (super-Turing)** Problems that have *mathematical* solutions but are not solvable by any TM. These include problems that are neither recursively enumerable nor recursively computable. They lie beyond the scope of classical computation.
- RE (recursively enumerable)** Also called *semi-decidable*. A TM can recognize solutions: if an input belongs to the language (*i.e.*, the answer is “yes”), the machine will eventually accept it. However, if the answer is “no,” the machine may run forever. Thus, non-membership is not decidable.
- R (recursively computable)** Also called *decidable*. Problems for which a TM can decide in finite time whether a given input belongs to the language. These problems can be both accepted and rejected by some algorithm.
- P (deterministic polynomial time)** Problems that can be solved by a deter-

ministic TM in polynomial time. These are considered “efficiently solvable.”

NP (non-deterministic polynomial time) Problems for which a candidate solution can be *verified* in polynomial time by a deterministic TM. It is unknown whether every problem in NP can also be *solved* in polynomial time (i.e., whether $P = NP$).

NP-complete The hardest problems in NP; if any one of them can be solved in polynomial time, then all NP problems can. Each NP-complete problem is both in NP and NP-hard.

PSPACE (polynomial space) Problems solvable by a TM using polynomial *memory*, regardless of how much time it takes. This class includes all of P and NP, but may also contain problems harder than NP.

The complexity classes 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 TM.

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 both number and precision. Nobody can see how Nature or human thought being super-Turing could ever be *proved*, so it is often held that any physical process can be effectively simulated by a TM; this is the *physical* Church-Turing thesis (Aaronson, 2005; Copeland, 2024). Compositional algorithms that halt are NP or less.

Whether or not NP is contained in P is one of the most important open questions in science. Most mathematicians and scientists believe, for overlapping reasons, that $P \neq NP$. If $P = NP$ can be proved, it becomes possible in principle to automatically solve all problems of a size that human beings can grasp. But if $P \neq NP$ can be proved, we will know that some problems are too complex for computers to solve.

2.2. Algorithmic Composition in Context

We can now return to the analysis of algorithmic composition with more understanding. Some compositional algorithms use strictly *finitary* methods, and others use implicitly *infinitary* methods. Both finitary and infinitary algorithms can exhibit *computational irreducibility* in the sense of Wolfram (Wolfram, 1985). In order to know what any non-trivial algorithm actually does, one must actually run the algorithm.

Examples of finitary algorithms include automatic counterpoint generation based on Renaissance rules (Schottstaedt, 1989; H. Taube, 1991), generating scores using finite iterations of Lindenmayer systems (McCormack, 1991), and so on. Gwee formalized algorithms for constructing counterpoint given a *cantus firmus*, constructed decision problems for those algorithms, and proved they are in PSPACE (Gwee, 2002, 2013). This implies that rule- or constraint-based compositional problems are as hard as *any* recursively computable problem.

Examples of infinitary algorithms include stochastic music computed with real-valued random variables (Xenakis, 2001), music computed using real-valued dynamical systems (Beyls, 1991; Gardner, 1978; Voss & Clarke, 1975), and my Mandelbrot set/Julia set and IFS ideas.

Braverman proved that hyperbolic Julia sets are computable, and in P (measure 1); most parabolic Julia sets are computable, but in NP or worse (measure 0); and some Julia sets with Siegel discs are neither hyperbolic nor parabolic, but *uncomputable*

(measure 0) (Braverman, 2008; Braverman & Yampolsky, 2006, 2009).

ST Some Julia sets with Siegel discs might be musically interesting in the abstract, but they are uncomputable. Picking Julia set parameters at random won't find any of these.

RE The Mandelbrot set, properly speaking, is not *recursively computable*, *i.e.* not Turing computable (Blum & Smale, 1993). Plots we make of the set are approximations. 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.

NP If some parabolic Julia sets are musically interesting, then computing a parametric map that is complete enough to be useful for composers might remain forever computationally intractable. For, as Julia set parameters are chosen closer and closer to a Siegel point, the Julia sets get harder and harder to compute, and the appearance of computer generated plots or scores fluctuates wildly. It is not clear whether **NP** includes any large number of musically interesting compositions.

P But if $P = NP$ then in principle it is possible to produce a useful parametric map in polynomial time: no sets with Siegel discs or Cremer points, but all hyperbolic points and as close as we like to almost all parabolic points. In that case, parametric composition might open up a vast new world for composers.

Julia sets are not the only attractors of dynamical systems that can be **ST**. Other types of dynamical systems that can have **ST** attractors include shift maps (Moore, 1991), piecewise affine maps (Bazille, Bournez, Goma, & Pouly, 2018), and continuous time recurrent neural networks (Postlethwaite, Ashwin, & Egbert, 2025). IFSs are recursively enumerable, but certain questions concerning their attractors are uncomputable (Dube, 1993). The literature is scattered. The following are true for all these systems:

Universality The degree to which compositional algorithms are effectively universal depends on how hard it is to compute attractors for parameter points that approach a parameter point having an computable attractor. The harder that is, the larger the intractable areas near the uncomputable points become, and the fewer pieces can be computed.

Mappability Only the ability to correlate numerical parameters for attractors with musical features of those attractors enables parametric composition. The degree of mappability depends on the computational complexity of attractors, especially those generated from parameters near the uncomputable points.

Tractability If $P = NP$, then a high-resolution parametric map of considerable subsets of compositions would 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 (almost) all possible structure in music, and partly overcome the irreducibility of compositional algorithms. In other words, $P = NP$ could imply that algorithmic composition can be made *intelligible*.

But if $P \neq NP$, parametric maps must exclude not only points with uncomputable attractors, but also larger neighborhoods of these points. This would to some extent fracture and obstruct the intelligibility of parametric composition.

The reader may well be wondering why I am talking about uncomputability when algorithmic compositions are normally *computed*. The reason is, composers normally don't think about finitary or infinitary methods, or about computability or uncomputability. Indeed, composers often seem to simply assume continuous mathematics using real numbers. But all implicitly infinitary ideas must of course be implemented,

using finitary methods, on a physical computer.

We should always remember: things may be mathematically true, yet not computable; other things may be computable in the abstract, yet not computable in our physical world; and still other things may be physically computable, yet only given resources far exceeding ours.

Beneath this issue is the older issue of philosophical realism (*e.g.* considering even uncomputable Julia sets to be real in the Platonic sense, which ultimately derives from Pythagoreanism) versus pragmatism or nominalism (considering uncomputable Julia sets a convenient fiction). With respect to music, realism implies that a composer’s idea is based on their grasp of an abstract yet real musical object. With respect to algorithmic composition, realism implies that pieces based on uncomputable attractors are nevertheless real musical objects, and neglecting them is ignorant, while pragmatism and nominalism say “Just shut up and run the algorithm, the only pieces that matter are those we can compute.”

If neither can be proved, we will simply never know.

Another significant aspect of parametric composition in particular, and of algorithmic composition in general, is computational irreducibility. Almost every connected Julia set (measure 1) is the chaotic attractor of its generating recurrence. Therefore, the orbit of the Julia equation is computationally irreducible (Zwirn, 2015). The orbit of the equation cannot be determined by examining the equation, and cannot except in trivial cases be determined even by mathematically analyzing the equation. In order to know the orbit, it is necessary to actually run a program that computes the orbit. Even then, we can only obtain an approximation. What is true of Julia sets is true of IFSs and, as many researchers believe, of many other algorithms.

In sum, uncomputability demonstrates fundamental limitations of compositional algorithms, and irreducibility demonstrates the only partial intelligibility of tractable algorithms.

3. Methods of Algorithmic Composition

Before further exploring the mathematical foundations of algorithmic composition, I will provide more background on algorithmic composition software.

I define *algorithmic composition* as the use of computer software to write musical compositions that would be difficult or impossible to write without software. It does not include the use of notation software to write down music that one imagines (as that can be done with paper and pencil), nor the use of audio workstation software to record and overdub music that one improvises (as that can be done with a tape recorder). Algorithmic composition consists of all compositional techniques *idiomatic* to the computer. There are many such (Ariza, 2023; Fernández & Vico, 2013). A recent summary is (McLean & Dean, 2018). 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 (Glass, 2015; Potter, 2002), 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.

Irreducibility is a spectrum, not a binary choice. The minimum of irreducibility occurs when the composer simply writes down what they hear in their imagination. The maximum occurs when the composition is generated at random, so that there is 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) sounds different from the texture of another random variable (*e.g.* brown noise). In the middle of the spectrum is an area where the composer has *some* insight into the kind of music that will be generated, even though 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 & Isaacson, 1957) is the first piece that is unambiguously computer music. The suite is an algorithmic composition assembled using a toolkit of stochastic note generators and music transformation functions, as detailed in *Experimental Music* (Hiller & 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 like Open Music (Assayag, Agon, et al., 2004), Common Music (R. Taube, 2021, 2023), and many others. This is to date the most widely used method of algorithmic composition.

3.1. Algorithms in the Toolkit

The toolkit approach includes generators and transformations from many sources:

Traditional Music Theory Fugue and other canonical forms, scales, transpositions and rescalings of pitch and time.

“Atonal Theory” or “Set Theory” Transpose, invert, and reverse, and other group actions on tone rows or other pitch collections (Rahn, 1991)

Neo-Riemannian Theory Mathematizes voice-leading and discovers groups acting on notes, scales, and chords, together with their symmetries (Callender, Quinn, & Tymoczko, 2008; Tymoczko, 2006, 2011).

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

Dynamical Systems and Cellular Automata Borrowed from mathematics and used to generate series of musical properties such as pitches (Miranda, 1993).

Fractals Borrowed from mathematics and used to generate some or all aspects of musical scores (Madden, 2007; Miranda, 2001). Some composers, including myself, prefer to use an algorithm, such as a Lindenmayer system (Gogins, 1992a; Prusinkiewicz, 1986; Prusinkiewicz & Lindenmayer, 1996 [1991]) or IFS (Barnsley, 1993 [1988]; Gogins, 1991) that will generate an entire piece based on fractals or other mathematical methods, needing no further editing or assembling.

Evolutionary Computing Applying evolutionary computing to any of the above (Miranda & Biles, 2007).

3.2. Intrinsic vs. Extrinsic Algorithms

Most algorithms in the toolkit are computationally irreducible. However, there is a critical difference between applying, *e.g.*, a sequence of transformations from the General Contextual Group (Fiore & Satyendra, 2005), even if the transformations are selected at random, versus simply sampling a random variable to sequentially scramble or transpose a chord. I define as *intrinsic* all algorithms that act on primitive elements

of music and generate musically acceptable transformations of those primitives, and *extrinsic* all algorithms that are based on non-musical primitives, use transformations of the primitives that are not necessarily musically acceptable, or require mapping the output of the algorithm to musical scores.

In other words, while both procedures are computationally irreducible, any sequence of GCG actions will generate a much higher proportion of musically acceptable results than a sequence of random chords. That is because the GCG models some aspects of common musical practice. Other such syntaxes include abstracting the pragmatic rules of voice-leading, abstracting the network of chord progressions by scale degree in functional harmony, and so on.

Note well, a fractal or other mathematically derived algorithm may be modified to be based on, or to transform, musical primitives.

3.3. *Live Coding*

A more recent style of algorithmic composition known as *live coding* extends the toolkit approach. A live coding system consists of a library of routines that are assembled into a music-generating graph during a live performance by interpreting real-time commands in a domain-specific language. Such 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 be gleaned from the TOPLAP web site (toplap.org, 2023) and the *Oxford Handbook of Algorithmic Composition* (McLean & Dean, 2018).

3.4. *Artificial Intelligence*

Recently it has become possible to compose music using large language models (LLMs) like ChatGPT. This can be called the *artificial intelligence* (AI) approach to computer music. I discuss only LLMs as they are currently the most influential method of AI. Briefly, the approach is based on biology — 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 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 to learn weights indicating how tokens within the entire data set are related syntactically and semantically; each head looks at a different aspect of the data. The trained heads are then used to transform the input (the prompt) *plus* the current output (hence the term “*self*-attention”) into a new input, from which the next token of output is predicted (Vaswani et al., 2017). This attention mechanism, and other heuristics, 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 huge 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 AI, there are some reviews and critical studies of the capabilities and limitations of LLMs. From the skeptical side, see (Dale, 2021). For an amusing series of dialogues between all sides, see (Aaronson, 2023). This experience

makes it possible to identify some important limitations of AI:

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 of the LLM, because these are actually quite simple, but we have no real idea how an LLM like ChatGPT can conduct a fact-filled conversation with one in perfect English. The details are scattered through hundreds of billions of neural network weights in the LLM. Computational opacity goes far beyond computational irreducibility — 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 step by step; 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 LLMs too often generate factually incorrect responses to prompts. It is a reminder that the software has no way to compare its responses with the real world. I suspect hallucinations arise from human mistakes, conflicting goals, and outright lies 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 *we* have provided. This is a self-referential situation. When we converse with ChatGPT, we are looking at ourselves in a fun-house mirror.

4. Artistic Results and Procedures

To date, not many algorithmically composed pieces have become popular or influential, even among aficionados of art music and experimental music. A few paradigmatic pieces are Iannis Xenakis’ *La Légende D’Eer* (Xenakis, 2005) and *Gendy 3* (Xenakis, 1995), Charles Dodge’s *Viola Elegy* (Dodge, April 1994), and Brian Eno’s generative works (Eno, 2023; Eno & 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 software used. IRCAM has published a series of books with chapters by composers on their use of OpenMusic (Agon, Assayag, & Bresson, 2006a, 2006b, 2006-2016, 2016). These are very useful. Profiles of composers in *Computer Music Journal* are also useful. Here I outline 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 a 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 patterns on seashells or the branching of plants. I have used chaotic dynamical systems, Lindenmayer systems, IFSs, and other systems. And I have increasingly incorporated intrinsically musical primitives and transformations into, or along with, my algorithms (Gogins (2020), code at ?).

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 & Demers, 2021; Tu, Chen, Carlyn, & Chao, 2023). But actually this is another form of computational irreducibility, meaning in this case 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 the musical imagination* (see also Edwards (2011)).

Now the question arises, how can such opaque methods be used to compose good music? It is difficult but by no means impossible.

The parameters generally have a global effect on the final structure. For example, an IFS 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 for its sweet spot again.

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 (in LLMs, the loss function) at increasing levels of refinement (as with gradient descent).

Few if any algorithmic composers simply “hear music in their heads” and write software to render it. Most fool around producing various experimental chunks of music, refine them more or less as I have described, and assemble some into a finished composition.

5. Conclusion

Before I present my conclusions, I will summarize what I have learned about the mathematical foundations:

Uncomputability The set of musical compositions that are possible *in the abstract* is *recursively uncomputable*.

Universality In spite of the uncomputability of the set of all compositions, any *physically possible* composition is *recursively enumerable* or less.

- Irreducibility** Most compositional algorithms are *computationally irreducible*.
- Intrinsic vs. Extrinsic** Algorithms based on musical primitives and transformations produce a larger proportion of musically acceptable results.
- Opacity** Compositional algorithms based on AI are not only computationally irreducible, but also computationally opaque in that we have essentially no insight into the steps followed by the LLM.
- Mappability** Compositional algorithms are nevertheless always mappable. This ultimately is because sets of compositions can be ordered in some way by *musical* criteria, while their corresponding generating parameters can be ordered lexically or numerically.
- Intractability** Producing a useful parametric map of some subset of compositions is compute-intensive; applying complex rules to optimize a score can be in **PSPACE**.
- Hallucination** LLMs that are supposed to provide true or useful outputs sometimes just make stuff up. Hence material generated by an LLM in response to a prompt cannot be trusted. A person, indeed an expert, must evaluate the response. 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. LLMs can select, summarize, and vary — but cannot generate a response 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.

5.1. *Limitations*

At this time and for the foreseeable future, no form of AI is conscious or has its own goals; AI also is subject to hallucination and unoriginality. Therefore, for the foreseeable future, human composers must and will play an irreplaceable role in algorithmic composition. This involves selecting a set of possible compositions to study, evaluating the musical quality of each composition in the subset, and varying the parameters or prompts to improve the pieces.

Incomputability, intractability, opacity, and irreducibility 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.

5.2. *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 a parametric map for a fractal compositional algorithm. In all cases, starting with an initial trial, a final composition is approached by a descending, zigzag search through a multi-dimensional space representing musical possibilities of differing value, until the search comes to rest in some local optimum. This search can be greatly speeded up by using intrinsically musical algorithms that generate a larger proportion of musically acceptable results.

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 searching makes 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, due to concise commands and immediately audible feedback, provides faster searching. Spending time doing live coding also increases the composer’s insight into the tools.

5.3. *In Sum*

The major approaches to algorithmic composition — trial and error with a toolkit of algorithms, live coding, exploration of fractals, and AI — share the fundamental business of zigzagging down a slope in a multi-dimensional 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 parameter 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 NP or more.

Future developments in AI may have a significant impact on algorithmic composition. For example, AI 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 AI 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 seldom musically original; while algorithmic compositions based on toolkits, live coding, or fractals can be original, but are much more difficult to produce.

Usually the only way to penetrate the fog of incomputability, intractability, opacity, and irreducibility of the algorithms 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. 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:

- The increasing power of computer hardware, required to enable everything else.
- Speeding up the composer’s workflow, whether in parametric composition, algorithmic composition toolkits (*e.g.* Bellingham, Holland, and Mulholland (2019)), live coding, or AI.
- Defining more musically compact and intelligible spaces of musical possibility, based on a wider variety of algorithms that work with intrinsically musical primitives. This in my view is one of the most important tasks of future algorithmic composers: identify and refine such processes. However, it would be unwise to exclude extrinsic algorithms from the toolkit, as they might expand one’s hearing of what is musically acceptable.
- It may become feasible to compute a dense parametric map of a more or less compositionally universal algorithm in a reasonable amount of time. In fact, if $P = NP$ is ever proved, someone may succeed in creating an algorithm that

can compute a dense parametric map of an effectively universal algorithm in *polynomial* time.

The result might be, again, a God’s-eye view of possible structure in music. But I’m not holding my breath, and I think we must continue to work within a garden, it is to be hoped one that grows ever larger and richer, of disparate compositional algorithms.

References

- Aaronson, S. (2005). Np-complete problems and physical reality. *ACM SIGACT News*, 36(1), 30–52.
- Aaronson, S. (2023, February). *Should gpt exist?* Retrieved from <https://scottaaronson.blog/?p=7042> (Accessed: 2025-05-06)
- Agon, C., Assayag, G., & Bresson, J. (2006a). *The om composer’s book 1* (Vol. 2006). Editions Delatour France/Ircam-Centre Pompidou.
- Agon, C., Assayag, G., & Bresson, J. (2006b). *The om composer’s book 2. Editions Delatour/IRCAM Centre Pompidou, 2008.*
- Agon, C., Assayag, G., & Bresson, J. (2006-2016). *The om composers book.* (Available online at: <http://repmus.ircam.fr/openmusic/ombook>. Accessed 11 May 2023)
- Agon, C., Assayag, G., & Bresson, J. (2016). *The om composer’s book 3. Editions Delatour/IRCAM Centre Pompidou, 2016.*
- Ariza, C. (2023). *algorithmic.net.* (Available online at: <http://algorithmic.net/>. Accessed 13 May 2023.)
- Arora, S., & Barak, B. (2009). *Computational complexity: A modern approach.* Cambridge University Press. Retrieved from <https://theory.cs.princeton.edu/complexity/book.pdf>
- Assayag, G., Agon, C., et al. (2004). *Ircam openmusic.* (<http://freesoftware.ircam.fr>)
- Barnsley, M. F. (1993 [1988]). *Fractals everywhere* (2nd ed.). Boston: Academic Press Professional.
- Barnsley, M. F., & Demko, S. (1985). Iterated function systems and the global construction of fractals. In *Proceedings of the royal society of london a: Mathematical, physical and engineering sciences* (Vol. 399, pp. 243–275).
- Barnsley, M. F., & Sloan, A. D. (1989). Fractal image compression. In *Proc. scientific data compression workshop* (pp. 351–365).
- Bazille, H., Bournez, O., Gomaa, W., & Pouly, A. (2018, July). On the complexity of bounded time and precision reachability for piecewise affine systems. *Theoretical Computer Science*, 735, 132–146. Retrieved from <http://dx.doi.org/10.1016/j.tcs.2016.09.021>
- Bellingham, M., Holland, S., & Mulholland, P. (2019). Toward meaningful algorithmic music-making for non-programmers.
- Beyls, P. (1991). Chaos and creativity: The dynamic systems approach to musical composition. *Leonardo Music Journal*, 1(1), 31–36.
- Blum, L., & Smale, S. (1993). The gödel incompleteness theorem and decidability over a ring. In *From topology to computation: Proceedings of the smalefest* (pp. 321–339).
- Braverman, M. (2008). *Computability and complexity of julia sets* (Ph.D. thesis). University of Toronto, Toronto, Canada. (Advisor: Stephen Cook)
- Braverman, M., & Yampolsky, M. (2006). Non-computable julia sets. *Journal of the American Mathematical Society*, 19(3), 551–578.
- Braverman, M., & Yampolsky, M. (2009). *Computability of julia sets* (Vol. 23). Springer. (Available online at: <https://arxiv.org/pdf/math/0610340.pdf>. Accessed 11 May 2023))
- Callender, C., Quinn, I., & Tymoczko, D. (2008). Generalized voice-leading spaces. *Science*, 320, 346–348.
- Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., & Amodei, D. (2017). Deep reinforcement learning from human preferences. *Advances in neural information processing*

- systems*, 30.
- Copeland, B. J. (2024). The Church-Turing Thesis. In E. N. Zalta & U. Nodelman (Eds.), *The Stanford encyclopedia of philosophy* (Winter 2024 ed.). Metaphysics Research Lab, Stanford University. <https://plato.stanford.edu/archives/win2024/entries/church-turing/>.
- Dale, R. (2021). Gpt-3: What's it good for? *Natural Language Engineering*, 27(1), 113–118.
- Davis, M. (2018). *The universal computer: The road from leibniz to turing* (3rd ed.). Boca Raton, FL: CRC Press.
- Dodge, C. (April 1994). Any Resemblance is Purely Coincidental [Audio CD]. In (chap. Viola Elogy). New Albion Records.
- Dube, S. (1993). Undecidable problems in fractal geometry. *Complex Systems*, 7(6), 423–444. Retrieved from https://www.complex-systems.com/abstracts/v07_i06_a02/
- Edwards, M. (2011). Algorithmic composition: Computational thinking in music. *Communications of the ACM*, 54(7), 58–67. Retrieved from <https://dl.acm.org/doi/10.1145/1965724.1965742>
- Eno, B. (2023). *Generative music*. (Available online at: <https://inmotionmagazine.com/eno1.html>. Accessed 10 May 2023)
- Eno, B., & Chilvers, P. (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, J. D., & Vico, F. (2013). Ai methods in algorithmic composition: A comprehensive survey. *Journal of Artificial Intelligence Research*, 48, 513–582.
- Fiore, T., & Satyendra, R. (2005). Generalized Contextual Groups. *Music Theory Online*, 11(3).
- Gardner, M. (1978, January). Mathematical games: White, brown, and fractal music. *Scientific American*, 238(1), 16–32.
- Glass, P. (2015). *Words without music: A memoir*. WW Norton & Company.
- Gogins, M. (1991). Iterated Function Systems Music. *Computer Music Journal*, 15(1), 34–42.
- Gogins, M. (1992a). Fractal Music with String Rewriting Grammars. *News of Music*, 13, 146–170.
- Gogins, M. (1992b). ...how i became obsessed with finding a mandelbrot set for sounds. *News of Music*, 13, 129–139.
- Gogins, M. (2006). Score generation in voice-leading and chord spaces. In G. Essl & I. Fujinaga (Eds.), *Proceedings of the 2006 international computer music conference*. San Francisco, California: International Computer Music Association.
- Gogins, M. (2019). *More rant-like musings on algorithmic composition software*. (Available online at: <https://michaelgogins.tumblr.com/post/188345210028/algorithmiccompositionsystems>. Accessed 9 May 2023)
- Gogins, M. (2020). *Poustinia v6.7.2*. <https://youtu.be/gRocNenzwrs>. (Music composition on YouTube)
- Gogins, M. (In preparation). Parametric composition of score graphs.
- Gonsalves, R. A. (2021). *Ai-tunes: Creating new songs with artificial intelligence*. (Available online at: <https://towardsdatascience.com/ai-tunes-creating-new-songs-with-artificial-intelligence-4fb383>. Accessed 9 May 2023)
- Graham, L., & Demers, M. (2021). Applying neural networks to a fractal inverse problem. In *Recent developments in mathematical, statistical and computational sciences: The v amms international conference, waterloo, canada, august 18–23, 2019* (pp. 157–165).
- Gwee, N. (2002). *Complexity and heuristics in rule-based algorithmic music composition* (Ph.D. dissertation, Louisiana State University and Agricultural and Mechanical College, Baton Rouge, LA).
- Gwee, N. (2013). Music, mathematics, and microcomputers: Building upon a classical tradition. *GSTF Journal on Computing (JoC)*, 3, 1–6.

- Gödel, K. (1986). On formally undecidable propositions of *Principia Mathematica* and related systems i. In S. Feferman, J. W. D. Jr., S. C. Kleene, G. H. Moore, R. M. Solovay, & J. van Heijenoort (Eds.), *Kurt gödel: Collected works, volume i: Publications 1929–1936* (pp. 144–195). Oxford University Press. (Originally published in German in 1931)
- Hertling, P. (2005). Is the mandelbrot set computable? *Mathematical Logic Quarterly*, 51(1), 5–18.
- Hiller, L., & Isaacson, L. (Eds.). (1959). *Experimental music: Composition with an electronic computer*. New York, New York: McGraw–Hill.
- Huffman, C. (2024). Pythagoras. In E. N. Zalta & U. Nodelman (Eds.), *The Stanford encyclopedia of philosophy* (Spring 2024 ed.). Metaphysics Research Lab, Stanford University. <https://plato.stanford.edu/archives/spr2024/entries/pythagoras/>.
- Huffman, C. 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, T., & Kisaka, M. (2018). Julia sets appear quasiconformally in the mandelbrot set. *arXiv preprint arXiv:1804.00176*.
- Lei, T. (1990). Similarity between the mandelbrot set and julia sets. *Communications in mathematical physics*, 134, 587–617.
- Lejaren A. Hiller, J., & Isaacson, L. M. (1957). *Illiac suite for string quartet*. New York: New Music Editions.
- Madden, C. (2007). *Fractals in music: Introductory mathematics for musical analysis* (2nd, Revised and Expanded ed.). Salt Lake City, UT: High Art Press.
- Mandelbrot, B. B. (1982). *The fractal geometry of nature*. W. H. Freeman. Hardcover. Retrieved from <http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0716711869>
- McCormack, J. (1991). Grammar-based music composition using l-systems. In *Proceedings of the international computer music conference (icmc)* (pp. 484–487).
- McLean, A., & Dean, R. (2018). *The oxford handbook of algorithmic music*. Oxford University Press. Retrieved from <https://books.google.com/books?id=7XBGDwAAQBAJ>
- Miranda, E. R. (1993). Cellular automata music: An interdisciplinary project. *Interface*, 22(1), 3–21.
- Miranda, E. R. (2001). *Composing music with computers*. Oxford; Boston: Focal Press.
- Miranda, E. R., & Biles, J. A. (Eds.). (2007). *Evolutionary computer music*. London: Springer.
- Moore, C. (1991). Generalized shifts: unpredictability and undecidability in dynamical systems. *Nonlinearity*, 4(2), 199.
- Ocampo, R., Andres, J., Schmidt, A., Pegram, C., Shave, J., Hill, C., . . . Bown, O. (2023). Using gpt-3 to achieve semantically relevant data sonification 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* (pp. 212–227).
- OpenAI. (2023a). *Gpt-4 technical report*.
- OpenAI. (2023b). *Jukebox*. (Available online at: <https://openai.com/research/jukebox/>. Accessed 10 May 2023)
- Ord, T. (2006). The many forms of hypercomputation. *Applied mathematics and computation*, 178(1), 143–153.
- Patrick, E. A., Anderson, D. R., & Bechtel, F. K. (1968). Mapping multidimensional space to one dimension for computer output display. In *Proceedings of the 1968 23rd acm national conference* (pp. 511–515).
- Peitgen, H.-O., Jürgens, H., Saupe, D., Peitgen, H.-O., Jürgens, H., & Saupe, D. (2004). The mandelbrot set: Ordering the julia sets. *Chaos and Fractals: New Frontiers of Science*, 783–837.
- Piccinini, G., & Maley, C. (2021). Computation in Physical Systems. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philoso-*

- phy (Summer 2021 ed.). Metaphysics Research Lab, Stanford University. <https://plato.stanford.edu/archives/sum2021/entries/computation-physicalsystems/>.
- Postlethwaite, C. M., Ashwin, P., & Egbert, M. (2025). A continuous time dynamical turing machine. *IEEE Transactions on Neural Networks and Learning Systems*, 36(4), 6503–6515.
- Potter, K. (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* (pp. 455–457).
- Prusinkiewicz, P., & Lindenmayer, A. (1996 [1991]). *The algorithmic beauty of plants*. New York: Springer-Verlag. (Available online at <http://algorithmicbotany.org/papers/>)
- Rahn, J. (1991). *Basic atonal theory* (2nd ed.). New York: Longman.
- Reich, S. (2023). *Music as a gradual process*. (Available online at: <https://static1.squarespace.com/static/50e79ec7e4b07dba60068e4d/t/515707b0e4b0ec1768d61b17/130> Accessed 11 May 2023)
- Schottstaedt, W. (1989). Automatic counterpoint. In M. V. Mathews & J. R. Pierce (Eds.), *Current directions in computer music research* (pp. 199–214). MIT Press.
- Schwarz, K. R. (1980). Steve reich: Music as a gradual process: Part i. *Perspectives of New Music*, 19(1/2), 373–392. Retrieved 2023-05-11, from <http://www.jstor.org/stable/832600>
- Taube, H. (1991). Common music: A music composition language in common lisp and clos. *Computer Music Journal*, 15(2), 21–32. Retrieved from <https://doi.org/10.2307/3680360>
- Taube, R. (2021). *musx*. (Available online at: <https://github.com/musx-admin/musx>. Accessed 9 May 2023)
- Taube, R. (2023). *Common Music*. (<https://commonmusic.sourceforge.net/>)
- Tennant, N. (2023). Logicism and Neologicism. In E. N. Zalta & U. Nodelman (Eds.), *The Stanford encyclopedia of philosophy* (Winter 2023 ed.). Metaphysics Research Lab, Stanford University. <https://plato.stanford.edu/archives/win2023/entries/logicism/>.
- toplap.org. (2023). *Toplap*. (Available online at: <https://toplap.org/about/>. Accessed 14 May 2023)
- Tu, C.-H., Chen, H.-Y., Carlyn, D., & Chao, W.-L. (2023). Learning fractals by gradient descent. *arXiv preprint arXiv:2303.12722*.
- Tymoczko, D. (2006). The geometry of musical chords. *Science*, 313(5783), 72–74.
- Tymoczko, D. (2011). *A geometry of music: Harmony and counterpoint in the extended common practice*. OUP USA.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., . . . Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.
- Voss, R. F., & Clarke, J. (1975). 1/f noise in music and speech. *Nature*, 258(5533), 317–318.
- Wolfram, S. (1985). Undecidability and intractability in theoretical physics. *Physical Review Letters*, 54(8), 735.
- Xenakis, I. (1995). Xenakis: Ais, Gendy3, Taurhiphanie, Thallein / Nee, Steiger [Audio CD]. In (chap. Gendy3). Neuma.
- Xenakis, I. (2001). *Formalized music: Thought and mathematics in composition* (Revised edition ed.). Hillsdale, NY: Pendragon Press. (Originally published in French as *Musique Formelle*, 1963)
- Xenakis, I. (2005). *La légende d’eer* (G. Pape & I. Xenakis, Eds.). New York, NY: Mode. (“Electroacoustic work for 7-channel tape”—Container.)
- Zach, R. (2023). Hilbert’s Program. In E. N. Zalta & U. Nodelman (Eds.), *The Stanford encyclopedia of philosophy* (Spring 2023 ed.). Metaphysics Research Lab, Stanford University. <https://plato.stanford.edu/archives/spr2023/entries/hilbert-program/>.
- Zhang, C., Zhang, C., Zheng, S., Qiao, Y., Li, C., Zhang, M., . . . others (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, H. (2015). Computational irreducibility and computational analogy. *HAL*, 2015.