The Algorithm as the Score

Fabian S. Klinke

2026-08-29

on Brian Foo's Two Trains and data-driven generative music

At 1:37 in Brian Foo's 2015 work *Two Trains*, repeated attacks fill the upper register as the 2 train crosses the Financial District. At 3:53, between East 180th Street and Bronx Park East, far less sound surrounds a quiet pulse. Foo's program creates this contrast by turning census data and route distance into a sequence of samples (Foo, 2015).

In score-based music, composition usually lies in written notes, rhythms, instruments, dynamics and form. A digital audio workstation (DAW) already extends composition beyond the notes through choices about samples, presets, automation and the mix. Algorithmic generative music extends it again by using rules to produce musical events. This raises an awkward question: does the work exist in the code, the recording or the relation between them? *Medium theory* sharpens this question by treating software as part of the composition's form. Nick Collins treats the program and its sound as parts of one work (Collins, 2008).

Foo chooses the source data and sounds, then defines their relation in code. This *translation system* sets each section's order, length, instruments and volume before playback,

which makes its rules the composition. The recording realises the system as a performance realises a score. Writing the system and judging its results by ear are forms of *post-instrumental musicianship*.

Rules as scores

Rule-based composition predates computers. Generative music belongs to this wider tradition of rules that produce musical events. Michael Edwards traces such thinking from procedures for setting text and musical dice games to serialism and stochastic composition (Edwards, 2011). Software changes this practice by keeping the rule executable after the composer defines it. Through *remediation*, code takes over the score's role as a set of instructions and makes those instructions executable. A program can read structured data and apply the same calculation across a whole work without a person choosing each event.

Thor Magnusson argues that code can act as a score when it stores musical thought in a form that a computer can run (Magnusson, 2011). Computer-based data-driven generative music depends on this *executable form*. In Foo's system, Python reads route and income data, calculates the length and available instruments of each section, then writes a timed sequence that ChucK plays. The program applies the same mappings across all forty-eight sections. With a fixed input, it produces the same track each time. The generative work remains in the executable code behind that track (Foo, 2015).

A fixed score generated from a rule contains one result of that rule. Foo's program also retains the rule that joins data to sound. He can change an input or mapping, generate the whole route again and hear the new form (Foo, 2015). Code makes the relation between data and sound a reusable part of the composition.

The budget as a compositional system

The route creates the large-scale form. Foo divides the journey into forty-eight sections between forty-nine stations. Distance between stations sets each section's length, compressing a trip of about one hour and forty-five minutes into a four-and-a-half-minute track that retains the route's order and proportions (Foo, 2015). One rule composes duration across the whole journey.

Income changes what each section can sound like. Foo converts median household income from the 2011 American Community Survey into a monthly budget. Each section receives between three and thirty instruments, and the sounds become louder and more forceful as income rises. The Financial District section between Park Place and Chambers Street has an income of $205,192 and produces the loudest passage. East 180th Street has an income of $13,750 and produces the quietest (Foo, 2015). Income therefore controls the number of parts, their volume and their intensity.

In the code, income becomes a budget for sound. One table lists each station's income and location. A second gives every instrument a price, income bracket and musical rules. The function `buyInstruments` reads the instrument list from top to bottom. It skips sounds outside the station's bracket and buys an eligible one when the remaining budget allows. The function stops at the first eligible instrument that costs too much, even if cheaper sounds follow (Foo, 2015). The list turns income differences into access to sound. Its order, brackets and stopping rule define the program's *compositional affordances* by making some combinations possible and excluding others.

The program treats each station as a buyer, so a larger budget allows more musical parts. At 1:37, closely spaced attacks cover a broad register and keep the Financial District in

the foreground. At 3:53, a quiet pattern continues in the Bronx with little weight above it. These passages occupy very different amounts of space in the mix, and the code traces that contrast to the station budgets.

Foo's sample library mixes recordings by New York musicians with orchestral samples and a 2-train horn. He bases much of the pitch material on the subway door chime and models the pulse on Steve Reich's *New York Counterpoint*. Phase shifting suggests trains moving in and out of alignment (Foo, 2015). Foo chooses this musical image of New York before the algorithm arranges it.

Foo selects the dataset for the dramatic arc already present in its graph, which rises to a climax and then declines. After preparing the tables, he writes the Python program and changes its sounds and rules through repeated listening (Foo, 2015).

Data translation and authorship

Two Trains translates measured conditions into sound. Route coordinates become section lengths, and census income values become station budgets. Those budgets determine how many instruments a section can use and how strongly they play (Foo, 2015). Foo's algorithm turns a numerical description of New York into a sonic one by defining which values control each part of the music.

Computers make such translations repeatable at scale. Edwards describes algorithmic composition as a move from writing notes one by one to defining a process for the whole piece (Edwards, 2011). The composer must state each mapping clearly enough for the program to apply it across the dataset.

Stored in code, a mapping can operate beyond one dataset. It becomes a general system

for sonifying a phenomenon once the code defines how each measured feature changes the sound. New input with the expected structure produces another sonic account of the same phenomenon. A system may use one fixed dataset or connect the mapping to changing measurements. The translation rule persists across these outputs and forms the composed part of the system (Collins, 2008; Edwards, 2011).

Every reuse of the mapping carries the designer's decision about which differences matter in sound. The data is a selected numerical account of a real condition. A value may control duration, density or another musical feature. Sara Lenzi and Paolo Ciuccarelli argue that features of sound have no neutral relation to social values (Lenzi & Ciuccarelli, 2020). The mapping creates a *technocultural dialectic* by turning the designer's decisions into sound on every run.

Collins calls the design of such a process composition at a meta-level and argues that code can reveal musical causes hidden in a recording (Collins, 2008). He separates real-time generative works, where the program can count as the artwork, from offline algorithms that produce a score or recording. The fixed track places *Two Trains* in the second category.

The process shows *distributed agency*: Python and ChucK execute relations that Foo chooses and revises. The software acts within the limits he sets. Collins's wider method treats the program and its output as parts of one musical object. The translation rules perform the compositional work by governing the relation between possible inputs and outputs. Foo's production choices give the generated sequence its final sound.

Composition in the system

At 1:37, the Financial District fills the mix. The passage begins with a measured condition: the 2011 income value assigned to that part of the route. Foo's algorithm turns the value into a budget, then turns the budget into the number and force of the instruments. Before ChucK plays the sequence, the translation system has already placed the passage here and made it dense.

In rule-based data-driven generative music, the *translation system* is the composition. Medium theory locates this shift in the executable form of code. Its algorithm encodes measured features of a phenomenon as musical relations. Code preserves these relations so the composition can sonify later states. Each output gives one sonic account of a measured state. Within this post-instrumental practice, the musician defines a stable relation between source data and sound. Listening guides revisions, and the musician remains responsible for what the mapping means. To understand how such a work translates data, analysis must follow each output back through the code to the choices that define what measured differences should sound like. In *Two Trains*, Foo uses production to shape one fixed recording from the relation he composed in code.

References

Collins, N. (2008). The analysis of generative music programs. *Organised Sound*, *13*(3), 237–248. https://doi.org/10.1017/S1355771808000332

Edwards, M. (2011). Algorithmic composition: Computational thinking in music. *Communications of the ACM*, *54*(7), 58–67. https://doi.org/10.1145/1965724.1965742

Foo, B. (2015). *Two trains: Sonification of income inequality on the NYC subway.* https://datadrivendj.com/tracks/subway/

Lenzi, S., & Ciuccarelli, P. (2020). Intentionality and design in the data sonification of social issues. *Big Data & Society*, *7*(2). https://doi.org/10.1177/2053951720944603

Magnusson, T. (2011). Algorithms as scores: Coding live music. *Leonardo Music Journal*, *21*, 19–23. https://doi.org/10.1162/LMJ_a_00056