2020-08-10

Genetic Music System with Synthetic Biology

Miranda, Eduardo

10.1162/artl_a_00325
Artificial Life
Massachusetts Institute of Technology Press (MIT Press)

Accepted for publication in journal *Artificial Life* **26(3) -** Pre-publication draft

Genetic Music System with Synthetic Biology

Eduardo Reck Miranda

Interdisciplinary Centre for Computer Music Research (ICCMR)
University of Plymouth, Drake Circus
Plymouth PL4 8AA
United Kingdom

+44 1752 586 255

eduardo.miranda@plymouth.ac.uk

Abstract

This paper introduces GeMS, a system for music composition informed by Synthetic Biology. GeMS generates music with simulations of genetic processes, such as transcription, translation and protein folding, with which biological systems render chains of amino acids from DNA strands. The system comprises the following components: the Miranda Machine, the Rhythmator and the Pitch Processor. The Miranda Machine is an abstract Turing Machine-like processor, which manipulates a sequence of DNA symbols according to a set of programming instructions. This process generates a pool of new DNA strands, which are subsequently translated into rhythms. GeMS represents the musical equivalent of amino acids in terms of rhythms, referred to as rhythmic codons. This enables the Rhythmator to convert DNA sequences into rhythmic sequences. The Pitch Processor generates pitches for such rhythmic sequences. It is inspired by the phenomenon of protein folding. The Pitch Processor considers orientation information of DNA instructions yielded by the Miranda Machine in order to activate algorithms for generating pitches. A musical composition, entitled *Artibiotics*, for percussion ensemble and electronic instruments, is presented to demonstrate the system.

Keywords: Artificial Life music, genetic music, generative music, DNA music, Miranda Machine

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

1 Introduction

This paper describes a system for music composition whose generative mechanisms are informed by Synthetic Biology processes. A musical composition, entitled *Artibiotics*, is introduced to demonstrate the system.

Computing systems for synthesising novel proteins [1] and generative music software [2] rely on the processing of codes representing the realms of genetics and music, respectively. Notwithstanding the significant role played by the laws of biochemistry, academics in the field of Code Biology purport that the rules of genetic coding are largely arbitrary [3]. Similarly, despite substantial constraints imposed by the laws of auditory perception and cultural processes, the rules of music also are largely arbitrary [4]. It is remarkable that genetics and music systems share the characteristics of being symbolic, rule-based and generative. Hence the motivation to develop a system for creating music informed by Synthetic Biology.

By way of related work, there have been numerous initiatives to make music inspired or informed by Genetics. These range from generating music using neo-Darwinian models of evolution, known as Evolutionary Algorithms (EA), to methods for rendering DNA symbols into music and sound.

In a nutshell, those working with EA focus on developing systems to generate music inspired by neo-Darwinian processes by which biological organisms evolve [5, 6]. A typical example of an evolutionary music system is GenJam [7].

GenJam uses a Genetic Algorithm (GA) [8] to generate Jazz improvisations live on stage with a human performer. It maintains musical snippets in a database, which are retrieved for playback during an improvisation session. The retrieval is done according to set of fitness criteria established to best match the music that is played by the performer. GenJam applies various evolutionary operators to the musical snippets to breed new ones for future interactions. Another notable example is EvoBassComposer [9], which uses a GA to compose polyphonic music in the style of J. S. Bach's chorales.

Other approaches using EA deploy software agents programmed with specific music skills to interact with one another. Musical agents play sounds to each other and evolve shared musical vocabularies. Elements of the evolving vocabulary may be selected for reinforcement in their memories, discarded or modified, according to given constrains [10].

A typical example of converting DNA codes into musical notes is introduced by Temple [11]. Temple is interested in identifying properties of DNA sequences through sound. Also, Takahashi and Miller [12] developed a method to translate the DNA of proteins into music. Again, the authors are interested in using music to listen to patterns in DNA sequences. They propose music as an educational tool to introduce the general public and young children to Genetics. Another example is the system ComposAlign [13], developed to render large scale genomic data into music. This is a sophisticated system, which produces results in a format that can be uploaded into the Common Music [14] programming toolbox. This enables the user to explore different ways to listen to the results and compose pieces of music.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

In a more theoretical vein, Petoukhov [15] has been developing mathematical models to unveil symmetric properties shared by genetic systems and music, in particular musical scales.

The system introduced in this paper differs from evolutionary approaches as it does not use neo-Darwinian inspired EA to generate music. And it goes far beyond the usual practice of merely converting DNA sequences into music or sound.

What it being proposed here is a system for generating music by means of processes informed by Synthetic Biology. It includes a genetic machine, which generates pools of DNA strands from given genetic information, which are subsequently rendered into music. The music rendering algorithms are tightly coupled with the processes that produced the synthetic DNA strands.

2 Context and Basic Concepts

The work presented in this paper was developed in the context of an artist in residence programme supported by Biofaction, in connection with the Synpeptide project [16]. The Synpeptide project is aimed at the design of new antibiotics and involves a number of partners across Europe. The residency took place at Wagner Lab, in Regensburg, Germany, in 2017.

Essentially, Wagner Lab develops research in Synthetic Biology. The Lab is unravelling the structure and function of naturally occurring lantibiotics [17] with a view on engineering new kinds of lantibiotics. Lantibiotics are proteins produced by bacteria with antibiotic properties. In a nutshell, the research involves shuffling the DNA code of known lantibiotics, synthesise the new lantibiotics, and test them *in vivo* against specific kinds of harmful pathogens. This is an exceedingly laborious and time-demanding activity, as the number new possible re-combinations are immense. However, there are strategies for optimising this. For instance, given that lanbitiotics' codes follow a modular structure, where each module contributes a distinct characteristic of the molecule, the re-combinations can be done at the level of those modules. For instance, sections marked as A, B, C, D and E in the structure shown in Figure 1 are different modules of a lantibiotic known as Nisin.

The Nisin molecule is encoded with 102 DNA bases. However, the DNA strand does not fully characterize the actual molecule per se. It needs to be rendered into a chain of amino acids, which forms a small protein, referred to as a peptide. This process involves three stages: transcription, translation and post-translational modifications.

Each triplet of DNA, referred to as a codon, encodes one amino acid of the chain. In simplistic terms, firstly DNA is transcribed into RNA, which is subsequently translated into amino acids. For instance, Nisin is formed by a chain of 34 amino acids; represented by 34 codons.

Although there are over 500 different amino acids in nature, only 20 of them is encoded by means of DNA directly; see Table 1. What is important to understand here is that there are 64 possible DNA codons to encode these 20 basic amino acids, which means that most of them can be encoded by more than one codon. For instance, whereas Glutamine is encoded by CAA or CAG, Lysine is encoded by AAA or AAG.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

Figure 1 shows a schematic rendering of Nisin from its DNA code. The first amino acid is Isoleucine, which correspond to the ATT codon of the DNA strand. And the last amino acid is Lysine, corresponding to the last codon: AAA. Note that Dhb and Dha, do not have an associated DNA codon in Table 1. These two amino acids resulted from post-translational modifications, which are chemical reactions that take place after the translation.

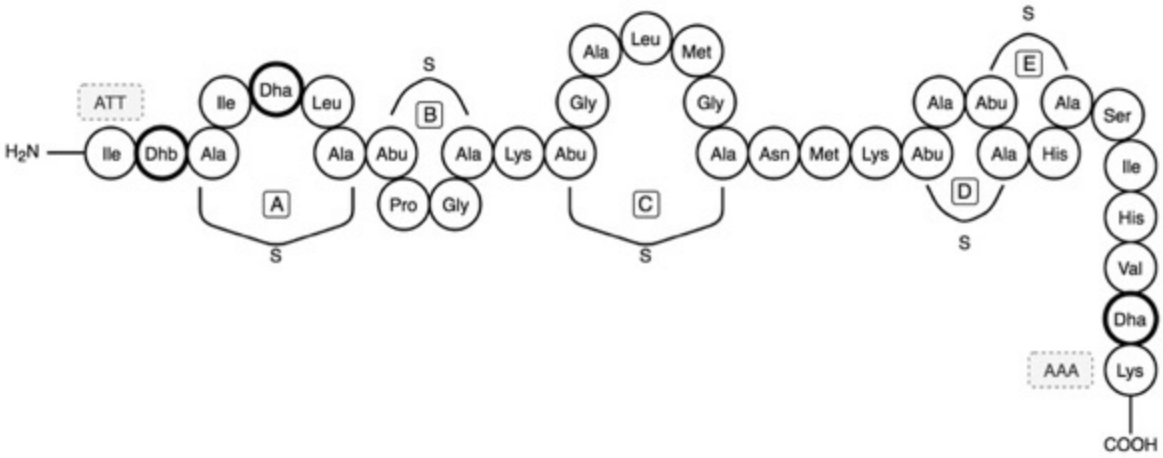

Figure 1: Structure of Nisin.

The more the author learned about the work developed at Wagner Lab, the more fascinated he became by the notion of synthesising new chimeric proteins. In many ways, the scientific modus operandi witnessed in the Lab reminded of his own creative modus operandi. It was felt that the process of synthesising new proteins is comparable to the way in which he composes music using computers [2]. Hence the idea of developing GeMS and the inspiration behind the composition *Artibiotics*.

3 Genetic Music System: GeMS

GeMS generates music with computer simulations of genetic processes with which biological systems render chains of amino acids from DNA strands to form proteins. At its most fundamental level a protein is coded as a strand of the DNA bases: Adenine, Guanine, Cytosine, and Thymine, abbreviated as A, G, C and T, respectively. A and G are referred to as Purines and C and T are referred to as Pyrimidines. Purines and Pyrimidines are considered as complements. Thus, A and T are complementary, and so are C and G.

A strand of DNA is parsed in groups of three consecutively occurring bases to produce an enzyme. In Biology, enzymes operate on DNA strands one base at a time to perform modifications on the strand. Essentially, amino acids are commands of a DNA programming language. Similarly, in GeMS each of those amino acids is associated with an instruction for an operation to be carried out on a DNA strand.

GeMS is informed by genetic processes of transcription, translation and protein folding. The system comprises the following components: (a) the Miranda Machine, (b) the Rhythmator and (c) the Pitch Processor.

Accepted for publication in journal *Artificial Life* **26(3) -** Pre-publication draft

At the core of GeMS is the Miranda Machine. Given a DNA strand, the Miranda Machine derives a program that processes the DNA strand itself, and generates new strands. That is, a DNA strand encodes instructions to modify itself and generate offsprings, which in turn modify themselves and so on.

The Rhythmator translates a given DNA sequence into a rhythmic sequence. GeMS represents the musical equivalent of amino acids in terms of rhythms, referred to as nucleo-rhythms. This enables the Rhythmator to parse a DNA strand and translate its codons into rhythmic codons.

The Pitch Processor is inspired by the phenomenon of protein folding. It considers orientation information of DNA instructions yielded by the Miranda Machine to 'fold' the derived program. The DNA instructions activate algorithms for generating pitches in function of their orientation.

3.1 Miranda Machine

The Miranda Machine builds upon a simple model of how a protein is manufactured, introduced by Hosftadter [18]. It is an abstract Turing machine-like processor [19] that manipulates a sequence of DNA symbols according to a set of rules.

A Turing machine is an abstract machine that manipulates a sequence of symbols according to a set of rules. By the same token, the Miranda Machine manipulates sequences of DNA strands: it transcribes a DNA strand into a sequence of instructions; that is, a DNA program.

Given a DNA strand, the Miranda Machine produces a chain of data processing instructions. In standard Biology this process would have produced a chain of amino acids, which would subsequently operate on DNA strands. In GeMS, the instructions forms a program that can process the originating DNA strand itself. Essentially, it transcribes a DNA code into a program to modify its own code.

The machine works as follows: firstly, it extracts codons (i.e., triplets of bases) from a given DNA sequence. These codons are used as indexes to retrieve instructions from a dictionary. Effectively, these are the instructions of the Miranda Machines' programming language to perform operations on DNA strands. For instance, the codon TCT retrieves the instruction 'move right by one unit'. Each instruction is represented by a symbol and is associated with one of the 20 canonical amino acids (Table 1).

In the aforementioned example, the instruction to 'move right by one unit' is represented by the symbol **movR** and is associated with the amino acid Serine. In Biology a specific amino acid can be represented by one or a group of different codons. Likewise, in GeMS, different codons may yield the same instruction. For instance, whereas AAA and AAG represent the amino acid Lysine, both codons also yield the instructions **delC**.

As the machine retrieves the instructions, the respective instructions are chained one after the other. Upon reaching the end of the DNA sequence, the resulting chain is a GeMS program. Subsequently, the GeMS program is applied to the very DNA sequence that served to generate it. The result will be one or more offspring DNA sequences.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

Amino acids	**Codons**	**Instructions**	**Orientation**	**Action**
Serine (Ser)	TCT, TCC, TCA, TCG, AGT, AGC	movR	S	Move right by one unit
Threonine (Thr)	ACT, ACC, ACA, ACG	movL	S	Move left by one unit
Alanine (Ala)	GCT, GCC, GCA, GCG	inA	S	Insert base A to the right
Glycine (Gly)	GGT, GGC, GGA, GGG	inC	L	Insert base C to the right
Isoleucine (Ile)	ATT, ATC, ATA	inG	L	Insert base G to the right
Leucine (Leu)	CTT, CTC, CTA, CTG, TTA, TTG	inT	R	Insert base T to the right
Proline (Pro)	CCT, CCC, CCA, CCG	addL	R	Insert random base to the left
Valine (Val)	GTT, GTC, GTA, GTG	addR	L	Insert random base to the right
Methionine (Met)	ATG	cop	L	Enable copy mode
Cysteine (Cys)	TGT, TGC	cut	S	Cut strand
Arginine (Arg)	CGT, CGC, CGA, CGG, AGA, AGG	swi	L	Switch machine head to another strand
Histidine (His)	CAT, CAC	delB	S	Delete base
Lysine (Lys)	AAA, AAG	delC	S	Delete codon
Asparagine (Asn)	AAT, AAC	PyR	L	Search for pyrimidine to the right
Aspartic acid (Asp)	GAT, GAC	PuR	R	Search for purine to the right
Glutamine (Gln)	CAA, CAG	PyL	R	Search for pyrimidine to the left
Glutamic acid (Glu)	GAA, GAG	PuL	R	Search for purine to the left
Phenylalanine (Phe)	TTT, TTC	invR	R	Invert base to the right
Tyrosine (Tyr)	TAT, TAC	invL	L	Invert base to the left
Tryptophan (Trp)	TGG	rev	S	Reverse the strand
	TAA, TAG, TGA	off	R	Disable copy mode

Table 1: The Miranda Machine's dictionary of instructions.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

In order illustrate how the Miranda Machine works, let us consider the DNA sequence shown in Figure 2 as the strand to be processed. It contains 45 bases, or 15 codons.

Figure 2: An example of a DNA sequence.

The sequence in Figure 2 yielded a GeMS program consisting of 15 instructions, as follows: { **cop**, **PyR**, **inA**, **PuL**, **swi**, **inG**, **cut**, **swi**, **rev**, **addL**, **off**, **invL**, **delB**, **movR**, **delC** }.

For example, ATG yielded **cop**, AAC yielded **PyR**, GCG yielded **inA**, and so on. If a DNA sequence finishes with an incomplete triplet (i.e., with 1 or 2 bases) then the system simply ignores it.

Before the Miranda Machine starts processing the sequence, it needs to establish a starting point for the GeMS program. For this example, it randomly selected an Adenine, which is indicated in bold in Figure 3.

Figure 3: The machine selected a random Adenine as a starting point.

In order to follow the next steps, imagine that the Miranda Machine has a processing head that reads one DNA base at a time as it processes the sequence. The head may move to the left or to the right.

The first instruction of the GeMS program is **cop**. This turns the machine into copy mode. In this case, the machine will make a copy of the respective base. And it will continue doing so as its head moves along the sequence until it is told to stop (with instruction the **off**).

Note that the base is always copied into their complement. That is, Adenine is copied into Thymine and Guanine is copied into Cytosine, and vice-versa. The result from applying **cop** is shown in Figure 4. As a convention, copies are placed above the original strand.

Figure 4: The result from applying **cop** onto the sequence shown in Figure 3.

Next is the instruction **PyR**, which tells the machine's head to move to the next Pyrimidine (i.e., C or T) on its right side. It found a T after taking four steps to the right. Note that as the head moved along the strand the machine continued to make copies of the visited positions. Metaphorically speaking, **cop** creates a mirror of the bases that are being copied. The result from this is shown in Figure 5.

Figure 5: The result from applying the instruction **PyR** on to the sequence in Figure 4.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

The instruction **inA** comes next, which tells the system to insert a base A on the right side of the head. The head moved to this new position and, of course, made a copy. The new result is shown in Figure 6.

Figure 6: The result from applying **inA** on to the sequence shown in Figure 5.

The full step-by-step run of the example above is given in Appendix 1. At the end of the run, the machine produced a set of five new DNA sequences:

1. CATATTGTCGCTGGCCTTAAGTATCATTCCAAA
2. TATG
3. GAGAGGCGGTA
4. C
5. TCC

In practice, the Miranda Machine is normally set to chain processing mode. That is, it continues processing the newly generated DNA sequences, which would subsequently generate new sets of sequences. It would carry on processing the new sequences, and so on, until a given halting criterion is met. The end result is a pool of DNA sequences, consisting of the originals and the groups of sequences derived from each respective cycle.

3.2 Rhythmator

The Rhythmator component translates a given DNA sequence into a rhythmic sequence. GeMS represents the equivalent of amino acids in terms of rhythms, referred to as nucleo-rhythms. The system holds vocabularies of four nucleo-rhythms, each of which is associated with one of the four bases A, T, G and C. The Rhythmator parses a DNA sequence and translate its codons into rhythmic codons.

In Biology, amino acids are classified in function to their chemical structure: Aliphatic, Aromatic, Acidic, Basic, and so on. Likewise, the Rhythmator holds different vocabularies of nucleo-rhythms for different types of amino acids.

Vocabularies of nucleo-rhythms are produced either manually or automatically. The system is able to extract rhythms from given musical scores and build the vocabularies. The nucleo-rhythms shown in Figure 7 were produced automatically from Brazilian samba and were attributed to Aliphatic amino acids. An explanation of the system that extract rhythms and builds vocabularies is beyond the scope of this paper; it would require detailed explanation of music segmentation techniques that are not relevant here.

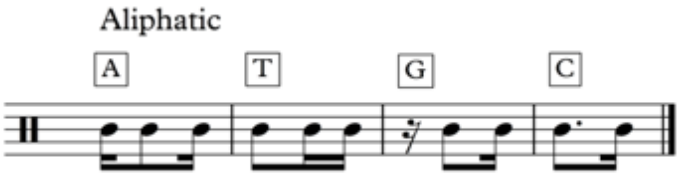

Figure 7: An example of a vocabulary of nucleo-rhythms for Aliphatic amino acids.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

Figure 8 shows examples of rhythmic codons for three Aliphatic amino acids (Alanine, Glycine and Isoleucine) built using the nucleo-rhythms shown in Figure 7. See Appendix 2 for an example of a complete set of vocabularies and respective set of rhythmic codons.

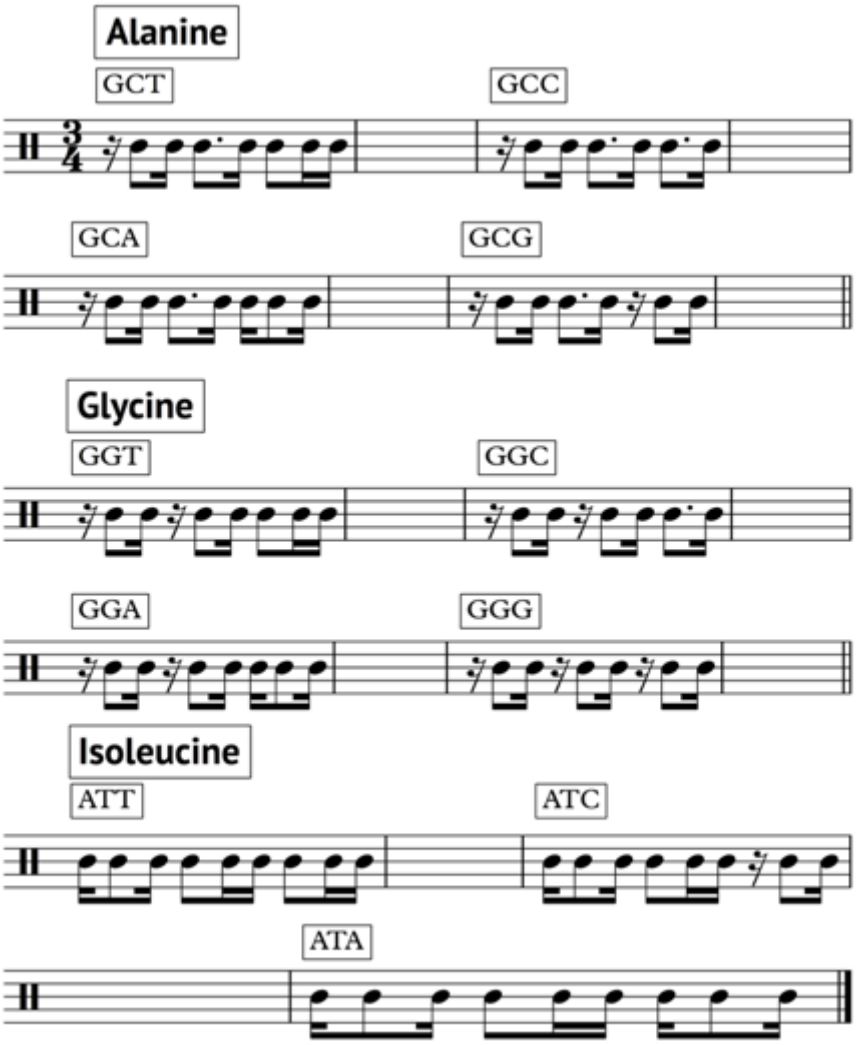

Figure 8: Rhythmic codons for three Aliphatic amino acids.

3.3 Pitch Processor

The Pitch Processor component is informed by the phenomenon of protein folding. Whereas in Biology protein folding defines the shape of a molecule, pitch folding in GeMS defines the shape of a GeMS program. However, whereas in Biology proteins are folded into three-dimensional shapes, here the shapes are two-dimensional. Ultimately, the shape of a program engenders agorihtms for generating pitches for a rhythmic sequence.

The fourth column of Table 1, gives orientation information for the instructions, referred to as RSL information. Pitch folding takes into account RLS information to fold a GeMS program.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

Each instruction is associated with one of three possible orientations: right, straight and left, represented as R, S and L, respectively. In order to visualize how the RSL information folds a program, let us represent each instruction as a block. Each block has an arrow pointing in the direction of its orientation. The examples shown in Figure 9 are the for commands **PuR**, **movR** and **swi**, with R, S and L orientations, respectively. The RSL information is notated in brackets.

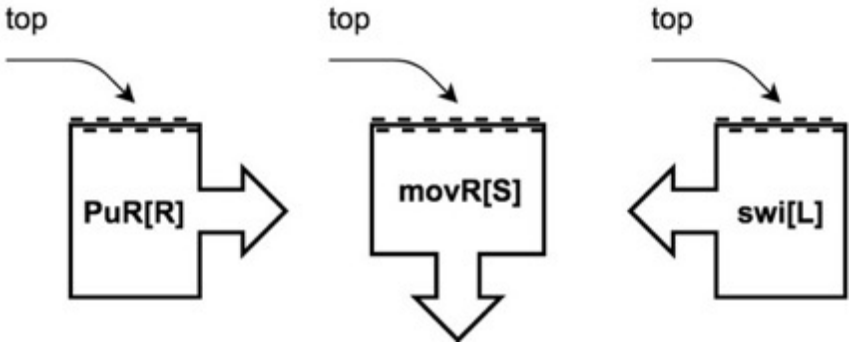

Figure 9: Visualisation of GeMS instructions as blocks with an arrow pointing in the direction of their respective orientations.

The Pitch Processor produces a domino-like chain of blocks, such that the arrow of a block always connects to the top of the next block, and so on (Figure 10).

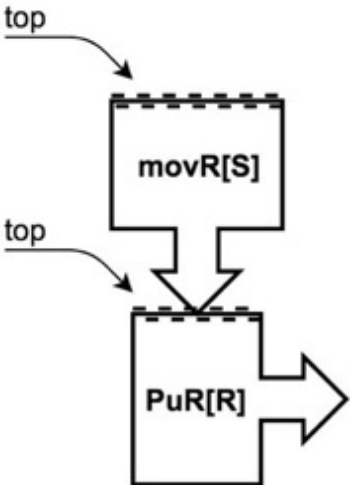

Figure 10: When connecting of two instructions blocks, each block is connected to the top of the next block.

As an example, consider the hypothetical strand TCTGGTGTACCCCAG. This strand yieldes a program with the following sequence of instructions and orientations: { **movR[S], inC[L], addR[L], addL[R], PyL[R]** }. Figure 11 depicts the visual block representation of the resulting folded structure.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

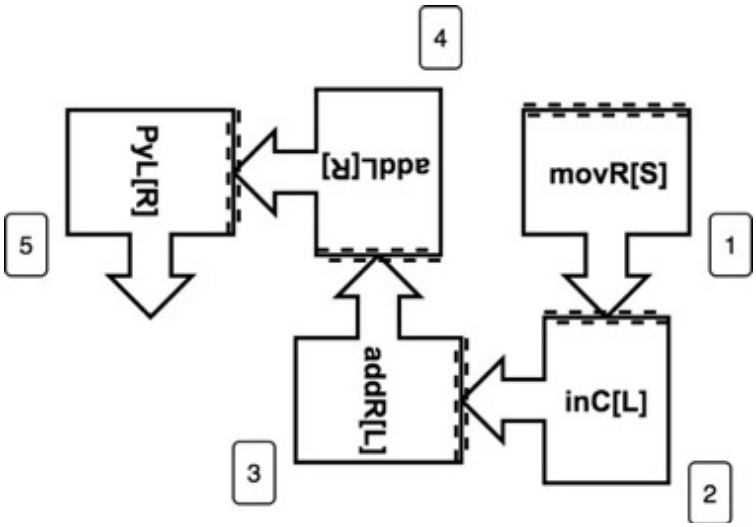

Figure 11: Visual representation of a folded program yielded by the sequence TCTGGTGTACCCCAG.

Once the program is folded, the Pitch Processor establishes the cardinal direction of each instruction in the block sequence and generates a respective cardinal expression.

There are four possible cardinal directions: north (NO), south (SO), east (EA) and west (WE). For instance, the arrow of the first block in Figure 11 is pointing to the south (i.e. downwards). Thus, **movR[S]** is re-written as **movR(SO)**. As another example, notice that the arrow of the third block is pointing to the north (i.e., upwards). Therefore **addR[L]** becames **addR(NO)**. Each instruction becomes cardinalized.

As a result, the cardinal expression of the folded program is expressed as { **movR(SO), inC(WE), addR(NO), addL(WE), PyL(SO)** } and is represented graphically as shown in Figure 12.

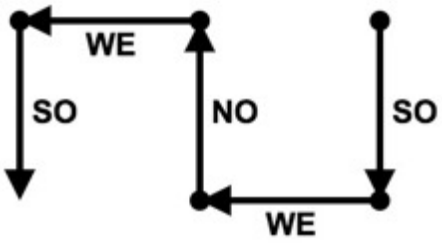

Figure 12: Abstract representation of the cardinal expression for the sequence TCT GGT GTA CCC CAG.

Each cardinalized instruction is associated with four distinct algorithms for generating pitches for a rhythmic sequence. Each cardinal direction follows a characteristic blueprint defining an abstract pitch sequence pattern. This is depicted in Figure 13. For instance, the WE blueprint generates pattern of the form {*s, u, d, s, u ,d, s, u, d, …*}. That is, given a starting pitch *s*, it produces a pitch above *s*, and then another pitch below *s*, and so on. And as another example, SO generates a sequence of the form: {*s, d, s, d, s, d, s, d, …*}. The actual number of semitones the next note will jump upwards or downwards is calculated in a number of different ways, depending on the instruction. For instance, the distances between ascending notes for **dec(NO)** are calculated differently than those for **int(NO)**.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

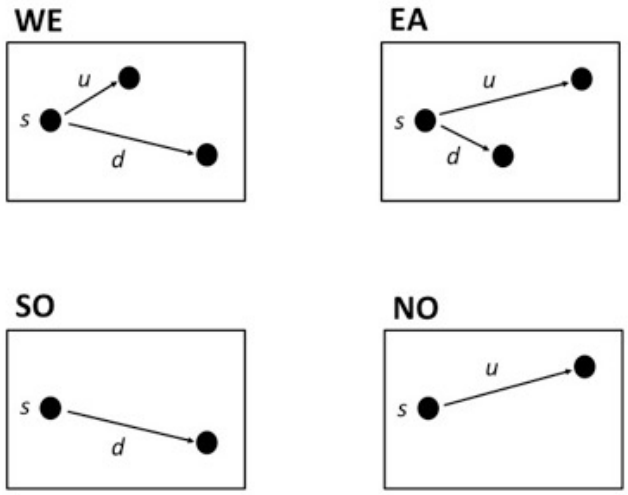

Figure 13: Cardinal blueprints for generating pitches for rhythmic sequences.

An example of four generative pitch algorithms for a cardinalized **swi** is given in Table 2 and for **dec** is shown in Table 3.

Instruction	Cardinality	Cardinal Algorithm
swi	WE	Given a reference pitch, transpose it 7 semitones downwards to generate pitch ***s*** Then: ***x*** = 1 Next: for each note, do while there are notes to process { assign pitch ***s*** to respective note calculate pitch ***u*** = ***s*** + ***x*** assign pitch ***u*** to respective note calculate pitch ***d*** = ***s*** – ***x*** assign pitch ***d*** to respective note ***x*** = ***x*** + 1 }
	EA	Given a reference pitch, transpose it 7 semitones downwards to generate pitch ***s*** Then: ***x*** = 1 Next: for each note, do while there are notes to process { assign pitch ***s*** to respective note calculate pitch ***d*** = ***s*** – ***x*** assign pitch ***d*** to respective note calculate pitch ***u*** = ***s*** + ***x*** assign pitch ***u*** to respective note ***x*** = ***x*** + 1 }
	NO	Given a reference pitch, transpose it 7 semitones downwards to generate pitch ***s*** Then: ***x*** = 1 Next: for each note, do while there are notes to process { assign pitch ***s*** to respective note calculate pitch ***u*** = ***s*** + ***x*** assign pitch ***u*** to respective note ***x*** = ***x*** + 1 }
	SO	Given a reference pitch, transpose it 7 semitones downwards to generate pitch ***s*** Then: ***x*** = 1 Next, for each note, do while there are notes to process { assign pitch ***s*** to respective note calculate pitch ***d*** = ***s*** – ***x*** assign pitch ***d*** to respective note ***x*** = ***x*** + 1 }

Table 2: Cardinal algorithms for instruction **swi**.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

Instruction	Cardinality	Cardinal Algorithm
dec	WE	Given a reference pitch, transpose it 7 semitones upwards to generate pitch ***s*** Then: for each note, do while there are notes to process { assign pitch ***s*** to respective note calculate pitch ***u*** = ***s*** + 1 assign pitch ***u*** to respective note calculate pitch ***d*** = ***s*** – 1 assign pitch ***d*** to respective note ***s*** = ***d*** }
	EA	Given a reference pitch, transpose it 7 semitones upwards to generate pitch ***s*** Then: for each note, do while there are notes to process { assign pitch ***s*** to respective note calculate pitch ***d*** = ***s*** – 1 assign pitch ***d*** to respective note calculate pitch ***u*** = ***s*** + 1 assign pitch ***u*** to respective note ***s*** = ***u*** }
	NO	Given a reference pitch, transpose it 7 semitones upwards to generate pitch ***s*** Then: for each note, do while there are notes to process { assign pitch ***s*** to respective note calculate pitch ***u*** = ***s*** + 1 assign pitch ***u*** to respective note ***s*** = ***u*** }
	SO	Given a reference pitch, transpose it 7 semitones upwards to generate pitch ***s*** Next: for each note, do while there are notes to process { assign pitch ***s*** to respective note calculate pitch ***d*** = ***s*** – 1 assign pitch ***d*** to respective note ***s*** = ***d*** }

Table 3: Cardinal algorithms for instruction **dec**.

In order to demonstrate how the Pitch Processor works, let us consider the rhythmic codon shown in Figure 14. And assume that this is an AGA codon. According to Table 1, AGA is an Arginine, which yields the instruction **swi**. Let us suppose that this instruction was cardinalized to WE: **swi(WE)**.

Figure 14: A hypothetic rhythmic codon.

As a reference for this example, the note C5 is used. Pitches are represented as MIDI pitch numbers [20]. The MIDI pitch number for note C5 is 72.

The pitch for the first note of the codon is created by transposing C5 by 7 semitones downwards: **s** = 72 – 7. This resulted in the MIDI pitch 65, which is a F4 note. Then, with ***x*** = 1, the pitch for the second note is calculated as ***u*** = 65 + 1. MIDI pitch 66 corresponds to a F#4 note. Subsequently, the third pitch is calculated as ***d*** = 65 – 1, which produced the note E4. Then, ***x*** = 2 for the next iteration of the algorithm. Therefore, the pitch for the forth note is ***s*** again. Then, the fifth pitch is calculated as ***u*** = 65 + 2, which results in G4. Finally, the sixth pitch was calculated as ***d*** = 65 – 2, which produces a D#4. The musical result is shown in Figure 15.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

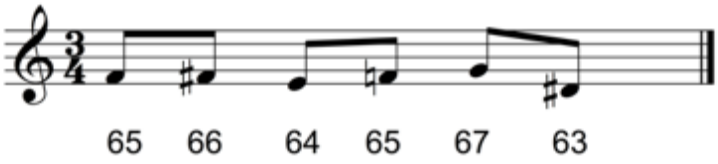

Figure 15: The resulting pitches for the AGA rhythmic codon. For reference, the respective MIDI pitch numbers are written below the notes.

4 From DNA Sequences to Musical Molecules

Let us glance over an example of how the components introduced above work together to generate a musical sequence; that is, a musical molecule.

Consider the DNA strand for a lantibiotic referred to as Columbicin:

GCTGGACGTGGATGGATTAAAACACTTACAAAAGATTGTCCAAATGTTATTTCATCAAT
TTGTGGAACAATTATTACAGCTTGTAAAAATTGTGCT

An initial run of the Miranda Machine produced the following set of DNA strands:

1. TCGTGTTAAAAATGTTCGACATTATCGCTACAAGGTGTTTAACTACTTTAT
   TGTAAACCTGTTAGAAAACATCGATGGGAA

2. AAATTAGGTAGGTGCAGGTCG

3. GC

The original strand (Columbicin) and the resulting strands are added to a DNA pool for further processing, either by the Miranda Machine again or by the other components of the GeMS system (Figure 21). Recall that the Miranda Machine is normally set to chain processing mode. That is, after an initial run, the machine goes on to process each of the results and adds the new outcomes to the pool. And this activity is repeated over and over again until a given halting criterion is met; e.g., a specific number of runs, or a certain target sequence is found.

For this example, let us run the machine only once, and pick the DNA strand number 2 to generate a musical molecule. The chosen sequence has 21 bases, or seven codons, as shown in Table 4:

Codon	Amino acid	Type	Instruction	Orientation
AAA	Lysine	Basic	delC	S
TTA	Leucine	Aliphatic	inT	R
GGT	Glycine	Aliphatic	inC	L
AGG	Arginine	Basic	swi	L
TGC	Cysteine	Sulfur	cut	S
AGG	Arginine	Basic	swi	L
TCG	Serine	Hydroxylic	movR	S

Table 4: The codons of the chosen DNA result and respective information.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

The Rhythmator is programmed to produce one bar of music per codon, in a ternary rhythm; that is, three beats per bar. Therefore, seven codons produce seven bars of music.

Considering the vocabularies and rhythmic codons provided in Appendix 2, the outcome from applying the Rhythmator to the strand resulted in the rhythmic sequence shown in Figure 16.

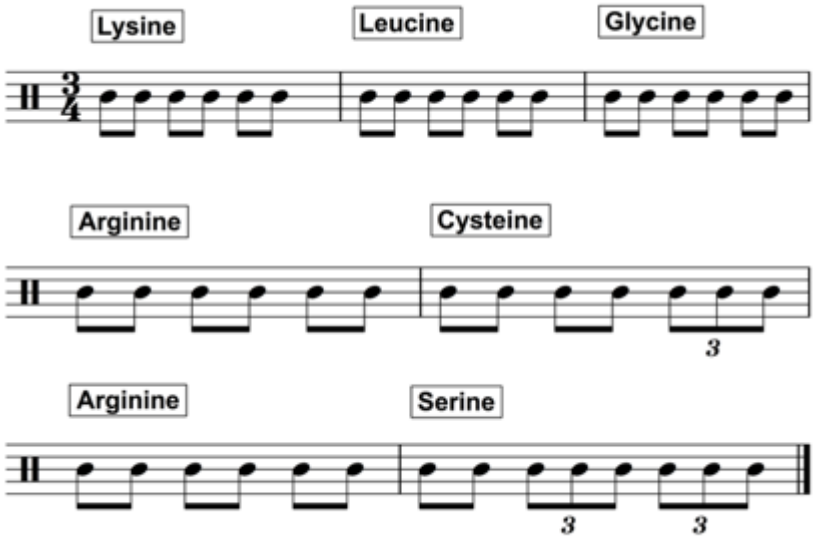

Figure 16: The rhythmic sequence yielded by the DNA strand number 2.

The Pitch Processor derives the following sequence of instructions and orientations: { **delC[S], inT[R]), inC[L], swi[L], cut[S], swi[L], movR[S]** }. The folded representation of the program is depicted in Figure 17.

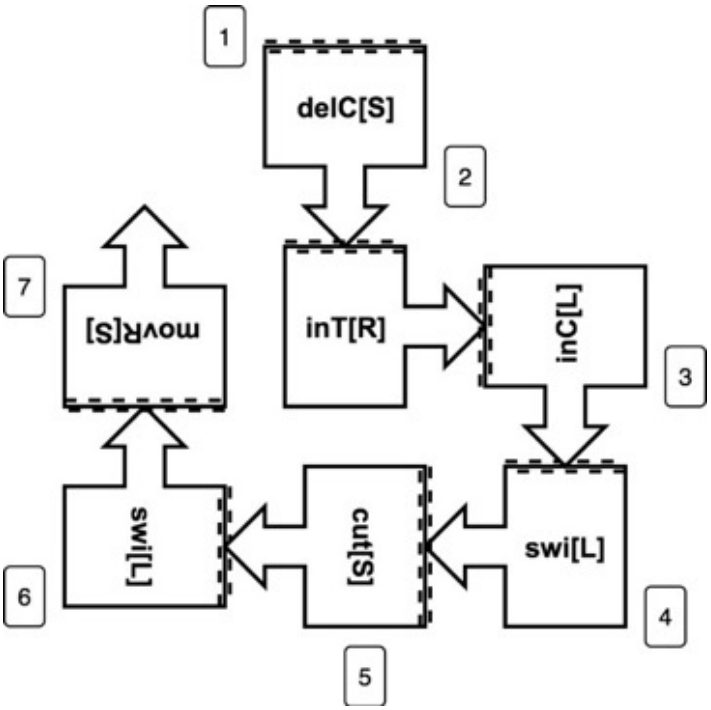

Figure 17: The folded program yielded by the DNA strand number 2.

Accepted for publication in journal *Artificial Life* **26(3) -** Pre-publication draft

Next, the system produces the following cardinal expression, which is represented graphically in Figure 18: { **delC(SO), inT(EA), inC(SO), swi(WE), cut(WE), swi(NO), movR(NO)** }.

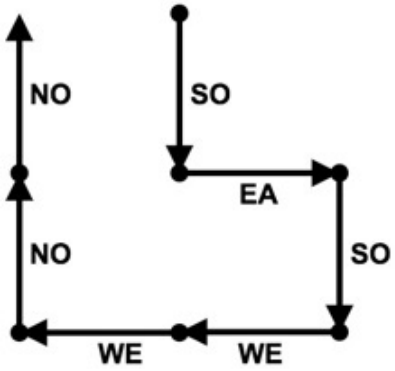

Figure 18: Cardinal expression for the sequence AAATTAGGTAGGTGCAGGTCG.

The respective cardinal algorithms were then applied to produce pitches for each bar of the rhythmic sequence in Figure 16. The result is shown in Figure 19. In this case the reference pitch for the algorithms was C4, MIDI number 60.

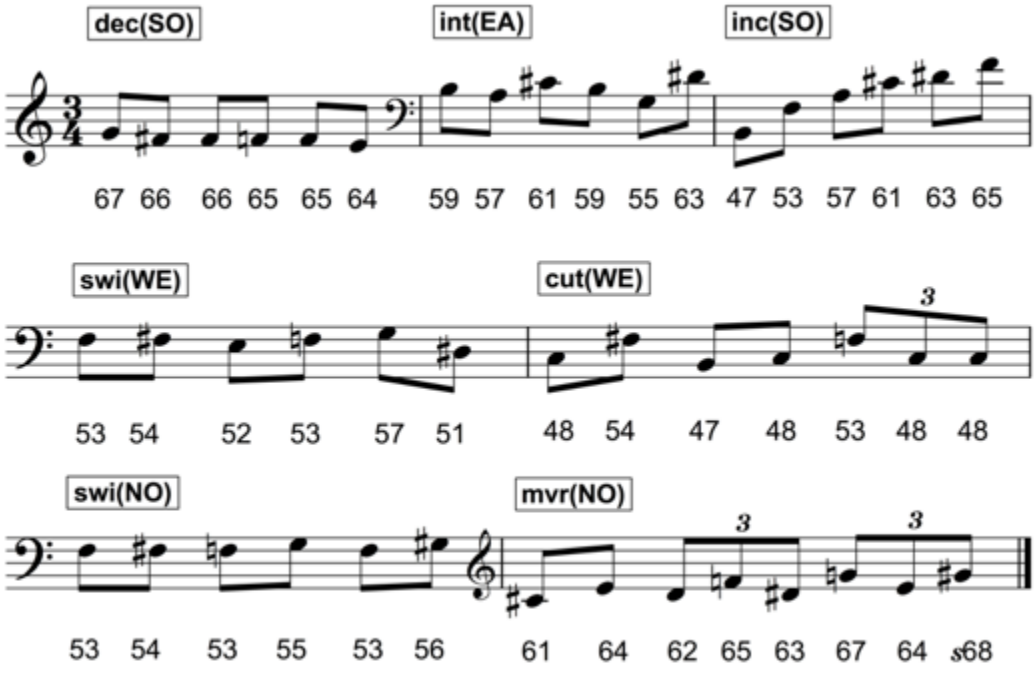

Figure 19: The resulting musical molecule yielded by the sequence AAATTAGGTAGGTGCAGGTCG. For reference, the respective MIDI pitch numbers are written below the notes.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

5 The Composition of *Artibiotics*

Artibiotics is a composition for percussion and electronic instruments. It is a metaphorical representation on an imaginary struggle to defeat a mutating pathogen using fictitious musical drugs. The drugs are composed with musical molecules generated from lantibiotics and articins. Articins are engineered proteins, which will be explained below. As the composition unfolds, various different drugs are trialled one after the other.

The pathogen is represented by musical parts played by a group of 25 percussion instruments. And the drugs are represented by musical parts played on electronic instruments. Samples of the musical score are shown in Appendix 3. The electronic parts are not fully notated on the score because they are pre-recordings, which are played back during the performance. On the score they are encapsulated as a box with wavy lines, as if they were a single instrument referred to as Electronics.

Articins are chimeric lantibiotics designed combining DNA modules from a number of lantibiotics; e.g., Nisin, Lactosin, Columbicin, and so on. For instance, below is Articin 5, consisting of 114 DNA bases, which was designed by combining DNA modules from Nisin and Lactosin. The bracketed portion of the Nisin code (shown in bold) replaced the bracketed portion of the Lactosin code to produce Articin 5:

Nisin: ATT ACA TCA ATT TCA CTT TGT ACA CCA GGA TGT AAA { **ACA GGT GCT CTT ATG GGA TGT** } AAT ATG AAA ACA GCT ACA TGT CAT TGT TCA ATT CAT GTT TCA AAA

Lactocin: TCA ACA CCA GTT TTA GCT TCA GTT GCT GTT TCA ATG GAA CTT CTT CCA ACA GCT TCA GT T CTT TAT { **TCA GAT GTT GCT GGA TGT** } TTT AAA TAT TCA GCTA AAC ATC ATT GT

Articin 5: TCA ACA CCA GTT TTA GCT TCA GTT GCT GTT TCA ATG GAA CTT CTT CCA ACA GC TTCA GTT CTT TAT **ACA GGT GCT CTT ATG GGA TGT** TTT AAA TAT TCA GCT AAA CAT CAT TGT

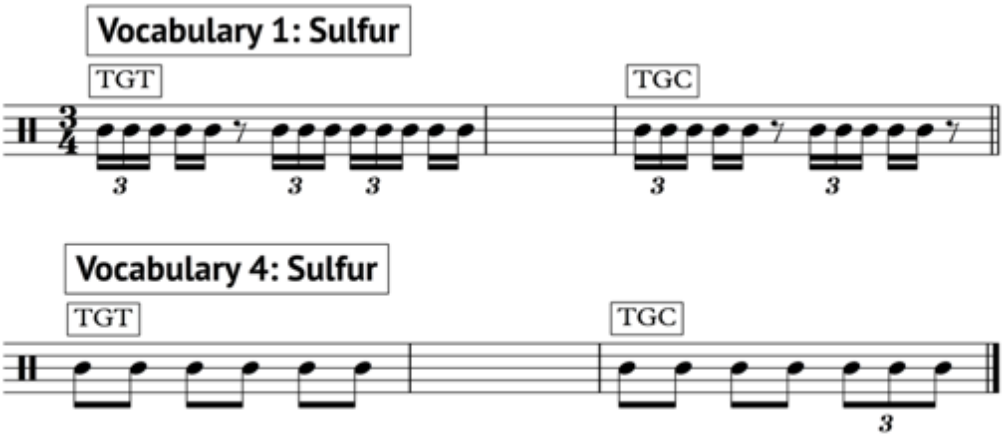

Figure 20: Examples from two different vocabularies of rhythmic codons.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

A total of 14 articins were made for the composition. These plus three known lantibiotics (Nisin, Lactosin and Columbicin) were used to compose 17 musical drugs.

Each musical drug was composed as follows: firstly, an articin (or lantibiotic) was processed by the Miranda Machine for 12 iterations. This resulted in a pool of new DNA strands. A total of 17 pools were generated for the piece. Then, the pools were processed by the Rhythmator to make rhythmic sequences.

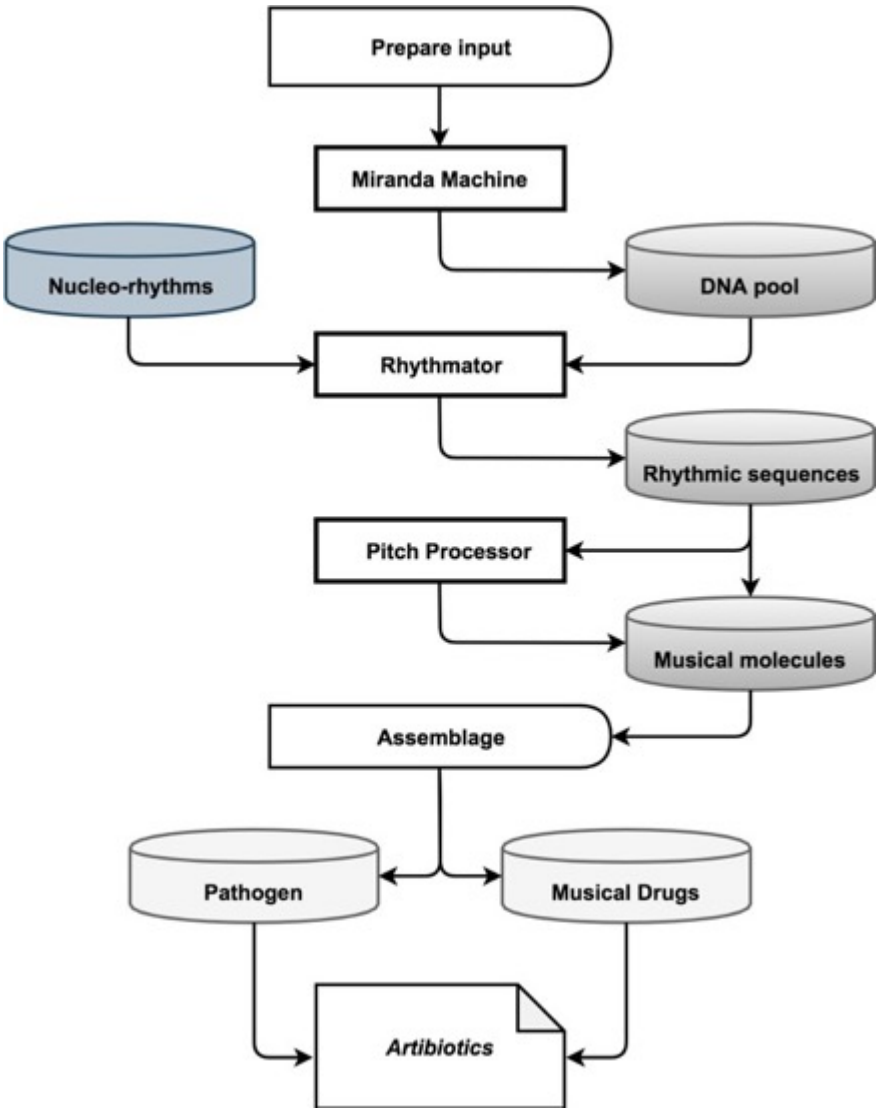

Figure 21: A schematic summary of the process of composing *Artibiotics* with GeMS.

Twelve different vocabularies of nucleo-rhythms were available for the Rhythmator so as to enable it to create varied types rhythmic sequences. These vocabularies were produced either manually or automatically from Brazilian samba music. From a musical perspective, it is useful to have different vocabularies available because musical compositions for a group of instruments require parts with different types of rhythms. For instance, in batucada (a substyle of samba characteristic of Brazilian Carnival), different percussion instruments (surdo, afuche, agogo, cuíca, and so on) play distinct rhythmic phrases. Examples from two contrasting vocabularies for two Sulfur rhythmic codons (TGT and TGC) are shown in Figure 20.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

The next step is to apply the Pitch Processor to generate pitches for rhythmic sequences. However, as this composition is for percussion instruments, it is important to bear in mind that some them are not melodic instruments. For instance, whereas one can play melodies on a marimba, it is not possible to play them on a drum. Therefore, sequences that were not processes by the Pitch Processor were also used in *Artibiotics*.

Once the pools of sequences were ready, then the musical drugs were assembled; one per pool. This process was done manually.

GeMS saves the results as MIDI files, which renders them compatible with production software routinely used by musicians. For *Artibiotics*, musical molecules were uploaded into Maschine 2, a piece of software manufactured by Native Instruments, which the composer used to assemble the drugs. The selection of molecules, the choice of the electronic timbres to play them, and other compositional decisions were made by the composer.

As for the composition of the pathogen, the DNA of the E. coli k-12 organism served as the initial input. The DNA of this organism contains over 4.5 million bases [21]. For practical reasons, a much shorter 1,000 long strand was extracted from the DNA to constitute the pathogen. The composition procedures were identical as for the composition of the drugs. The musical molecules were uploaded into Sibelius, which is the music notation software that was used to produce the score. Once uploaded, the molecules were assigned to different instruments of the percussion ensemble. Again, the selection of molecules, the allocations of instruments and other compositional decisions were made by the composer.

A schematic summary of the compositional process is illustrated in Figure 21. Audio examples are available in SoundClick [24]:

- Example 1: Articin 2.
- Example 2: Articin 5.
- Example 3: Articin 7. This can be heard together with the pathogen in Example 5.
- Example 4: Articin 13.
- Example 5: Columbicin.
- Example 6: Excerpt from the beginning of *Artibiotics* until end of section B. See scores in Appendix 3, Figures A3.1 and A3.2.
- Example 7: Excerpt from *Artibiotics*, from start of section N until the end of the piece. See scores in Appendix 3, Figures A3.3, A3.4 and A3.5.

6 Concluding Discussion

This paper introduced GeMS, a genetic music system for composition and a demonstration of a piece entitled *Artibiotics*.

An important stage of protein synthesis is missing in GeMS: the post-translational modifications (PTM) stage. PTM are chemical reactions that take place after translations. They produce amino acids which do not have any associated universal DNA codon (as listed in Table 1). In the example shown in Figure 1, Nisin's Dhb and Dha resulted from post-translational modifications.

There were a number of occasions in the process of composing *Artibiotics* where musical molecules had to be edited in order to be included in the piece; for instance, some contained

notes that were impossible to be produced by the allocated instrument. Metaphorically, these final adjustments are reminiscent of post-translational modifications. Future versions of GeMS will include a new component, which will be referred to as the Post-Translational Musificator. This component will modify musical molecules to match specific technical requirements (such as instrumentation) and also aesthetic ones.

It should be noted that GeMS was originally conceived with a particular way of composing music with computers in mind. That is, the machine is considered as a generator of materials and ideas to be further elaborated by a human composer. GeMS is not intended to replace human creativity. Rather, it is aimed at harnessing creativity.

Improvements to the system will not necessarily be aimed towards a system that composes music completely autonomously. Having said that, however, the Biofaction residency at Wagner Lab instigated a fascinating thought: Would it be possible to reverse the process? That is, would it be possible to engineer biological molecules from music? Would it be possible to synthesise a protein from, say, a Mozart piano sonata or a pop song? Would there be particular styles of music that might yield useful molecules? Could a piece of GeMS-generated music be reversed into the organism that yielded it? And if so, would it be possible to edit or embellish the music in meaningful ways in order to engineer new organisms? Of course, these are very broad speculations, but in theory they are not totally far-fetched as they may sound.

As a next step in this research, machine learning methods based on deep learning neural networks [22] combined with evolutionary algorithms [23] are currently being developed to enable the system to evolve its own set of instructions and algorithms. These would be constrained by the laws of Biochemistry. The ambition is to develop a machine with increased chances to produce biologically plausible molecules. Perhaps this might be the first step towards addressing the questions above.

Acknowledgements

The author is thankful to Markus Schmidt and Camilo Meinhart at Biofaction, and David Peterhoff at Wagner Lab, University of Regensburg, for their valuable support and advice. They engaged in enlightening discussions, which informed the various stages of this project. And many thanks to ICCMR's Satvik Venkatesh, whose computer programming skills were invaluable for the implementation of the system.

References

[1] Lorimer, D., Raymond, A., Walchli, J., Mixon, M., Barrow, A., Wallace, E., Grice, R., Burgin, A. & Stewart, L. (2009). "Gene Composer: Database software for protein construct design, codon engineering and gene synthesis", *BMC Biotechnology*, 9(36).

[2] Miranda, E. R. (2001). *Composing Music with Computers*. Oxford: Focal Press.

[3] Barbieri, M. (2018). "What is code biology", *BioSystems*, 164(2018), 1-10.

[4] Spencer, P. & Temko, P. M. (1994). *A practical approach to the study of form of music*. Waveland Press Inc.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

[5] Miranda, E. R. (Ed.) (2011). *A-Life for music: Music and computer models of living systems*. Middleton, WI: A-R Editions.

[6] Miranda, E. R. & Biles, J. A.(Eds.) (2007). *Evolutionary computer music*. Heidelberg: Springer.

[7] Biles, J. A. (1994). "GenJam: A genetic algorithm for generating Jazz solos", *Proceedings of the International Computer Music Conference (ICMC 1994)*, Aarhus, Denmark, September 12-17, pp. 131-137.

[8] Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. Cambridge, MA: The MIT Press.

[9] De Prisco, R., Zaccagnino, G. & Zaccagnino, R. (2010). "EvoBassComposer: a multi-objective genetic algorithm for 4-voice compositions". In M. Pelikan and J. Brawnke (Eds.), *Proceedings of the 12th Annual Conference of Genetic and Evolutionary Computation* (pp. 817-818). New York, NY: Association for Computing Machinery.

[10] Miranda, E. R., Kirby, S. & Todd, P. (2010). "On computational models of the evolution of music: From the origins of musical taste top the emergence of grammars", *Contemporary Music Review*, 22(3), 91-111.

[11] Temple, M. D. (2017). "An auditory display tool for DNA sequence analysis", *BMC Bioinformatics*, 18(221).

[12] Takahashi, T. & Miller, J. H. (2007). " Conversion of amino-acid sequence in proteins to classical music: search for auditory patterns", *Genome Biology*, 8(405).

[13] Ingalls, T., Martius, G., Hellmuth, M., Marz, M. & Prohaska, S. J., (2009). "Converting DNA to music: COMPOSALIGN". In I. Grosse, S. Neumann, S. Posch, S. Schreiber, & P. Stadler (Eds.), *German conference on bioinformatics 2009* (pp. 93-103) . Bonn: Gesellschaft für Informatik.

[14] Taube, H. (1997). "An Introduction to Common Music", *Computer Music Journal*, 2(1), 29-34.

[15] Potoukhov, S. (2015). "Music and the modeling approach to genetic systems of biological resonances". In W. Hofkirchner & D. Roddorf (Eds.), *Proceedings of the International Society for Information Studies Summit Vienna 2015*. Vienna: IS4IS.

[16] *Synpeptide project.* Available at https://www.biofaction.com/portfolio/synpeptide/ (accessed May 2020)

[17] Willey, J. M. & van der Donk, W. A. (2007). "Lantibiotics: Peptides of diverse structure and function", *Annual Review of Microbiology*, 61, 477-501.

[18] Hosftadter, D. R. (1979). *Gödel, Escher, Bach: An Eternal Golden Braid*. London: Penguin Books.

[19] Petzold, C. (2008). *The annotated Turing: A guidedtTour through Alan Turing's historic paper on computability and the Turing Machine*. Hoboken, NJ: John Wiley & Sons.

Accepted for publication in journal ***Artificial Life*** **26(3) -** Pre-publication draft

[20] Huber, D. M. (2007). *The MIDI manual: A practical guide to MIDI in the project studio*. London: Routledge.

[21] Blattner, F., Plunket III, G., Bloch, C. A., Perna, N. T., Burland, V., Riley, M., Collado-Vides, J., Glasner, J. D., Rode, C. K., Mayhew, G. F., Gregor, J., Davis, N. W., Kirkpatrick, H. A., Goeden, M. A., Rose, D. J., Mau, B. & Shao, Y. (1997). "The complete genome sequence of *Escherichia coli* K-12", *Science* 277(5331), 1453-1462.

[22] Goodfellow, I., Bengio, Y. & Courville, A. (2017). *Deep learning*. Cambridge, MA: The MIT Press.

[23] Goldberg, D. E. (1989). *Genetic algorithms in search, optimisation and machine learning*. Boston, MA: Addison Wesley.

[24] *Artibiotics recordings*. Available at https://soundclick.com/Artibiotics (accessed May 2020).

Accepted for publication in journal *Artificial Life* **26(3) -** Pre-publication draft

Appendix 1: A Detailed Run of the Miranda Machine

Input strand: ATGAACGCGGAGAGGATTTGTCGCTGGCCTTAGTATCATTCCAAA

Derived DNA programming code: { **cop, PyR, inA, PuL, swi, inG, cut, swi, rev, addL, off, invL, delB, movR, delC** }

Figure A1.1: By default, the Miranda Machine starts on a randomly selected Adenine.

Figure A1.2: The instruction **cop** puts the system in copy mode, and makes a copy.

Figure A1.3: The instruction **PyR** moves to a Pyrimidine on the right. As the head moves, the bases are copied.

Figure A1.4: The instruction **inA** inserts a base **A** to the right of the head.

Figure A1.5: The instruction **PuL** searches for a Purine on the left.

Figure A1.6: The instruction **swi** switches the head of the machine to another strand.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

Figure A1.7: The instruction **inG** insert a base G on the right.

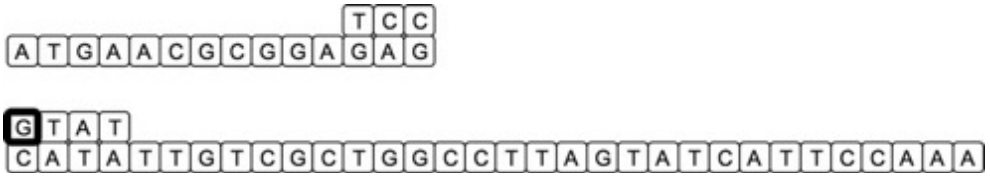

Figure A1.8: At this point, the instruction **cut** slices the coupled strands on the left of the head.

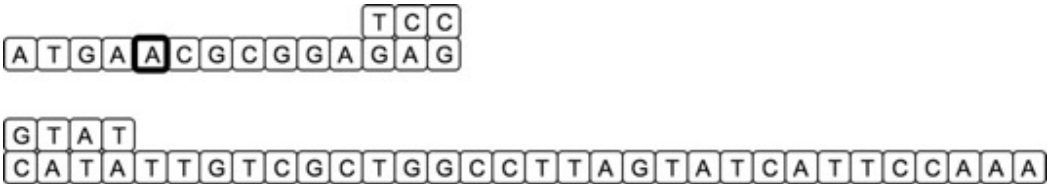

Figure A1.9: The instruction **swi** makes the head switches to another strand.

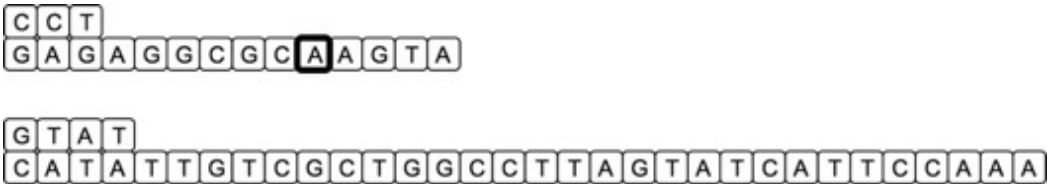

Figure A1.10: The active strand and its complement are reversed by instruction **rev**.

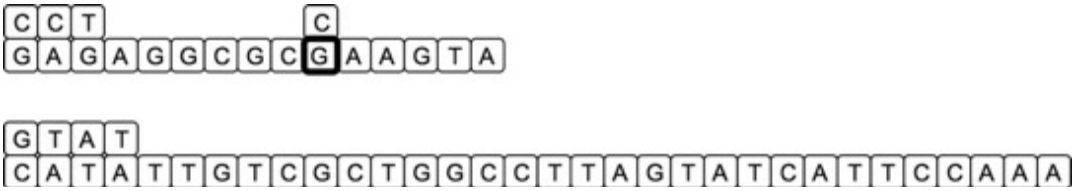

Figure A1.11: The instruction **addL** adds a random base, and its respective complement, on the left of the head. Subsequently, the instruction **off** disables the copy mode.

Accepted for publication in journal *Artificial Life* **26(3) -** Pre-publication draft

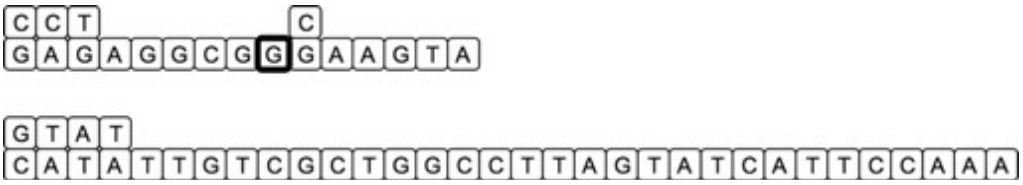

Figure A1.12: The base in the left of the head is inverted by instruction **invL**.

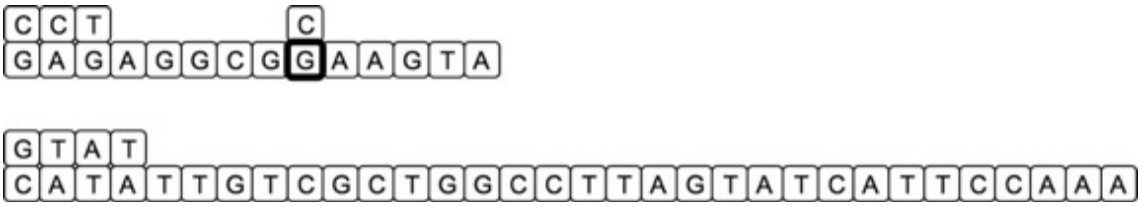

Figure A1.13: The instruction **delB** deleted a base.

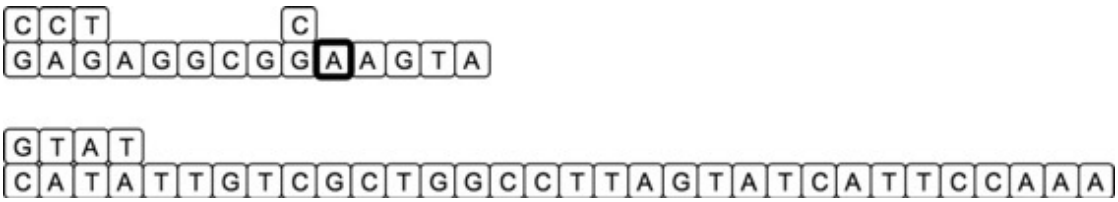

Figure A1.14: The instruction **movR** moves the head to the right by one unit.

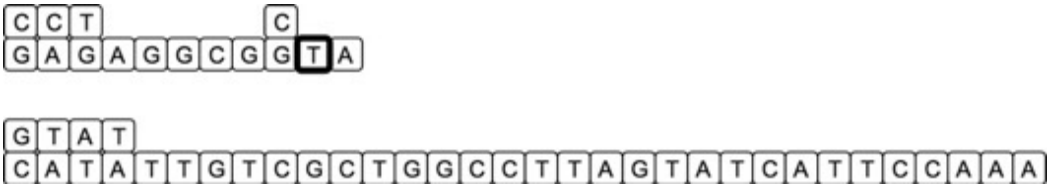

Figure A1.15: Finally, a codon is deleted with **delC**.

Accepted for publication in journal *Artificial Life* **26(3) -** Pre-publication draft

The resulting DNA strands are:

1. CATATTGTCGCTGGCCTTAAGTATCATTCCAAA
2. TATG
3. GAGAGGCGGTA
4. C
5. TCC

Note that copies of strands (the top rows) are written back to front.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

Appendix 2: Vocabularies and Rhythmic Codons

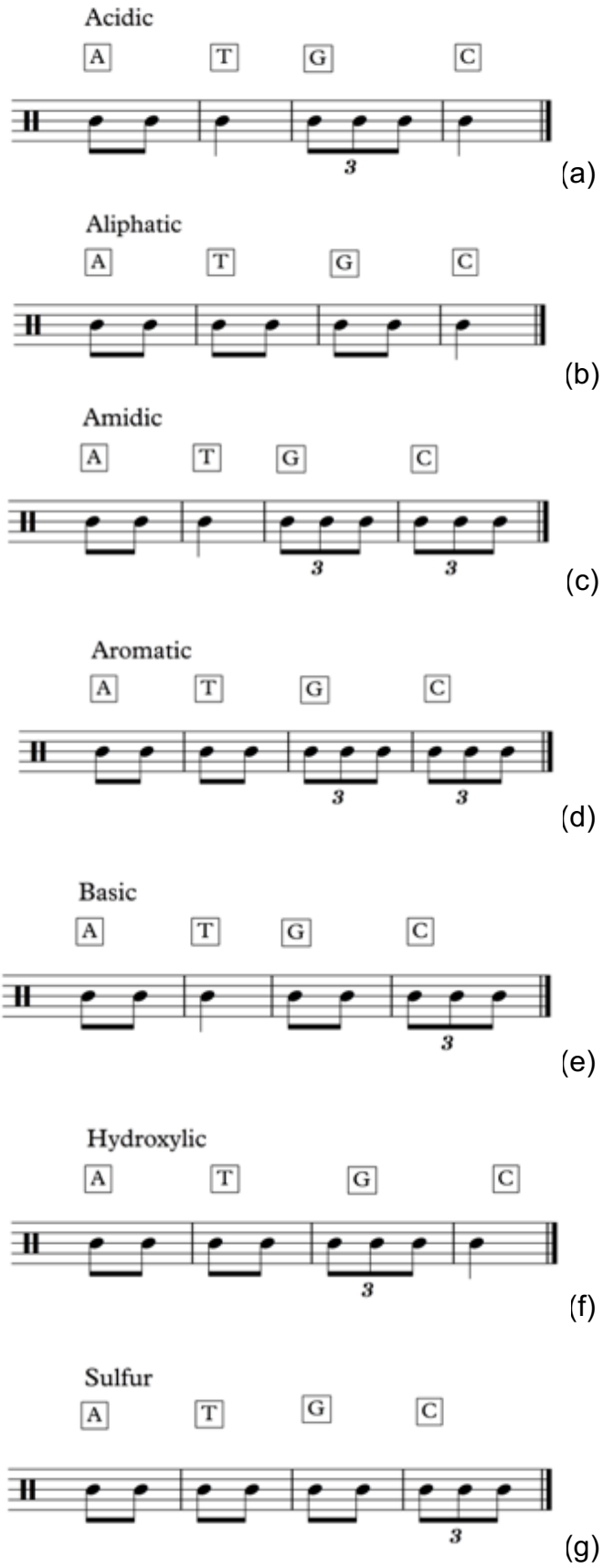

Figure A2.1: Vocabularies for seven types of amino acids.

Accepted for publication in journal *Artificial Life* **26(3) -** Pre-publication draft

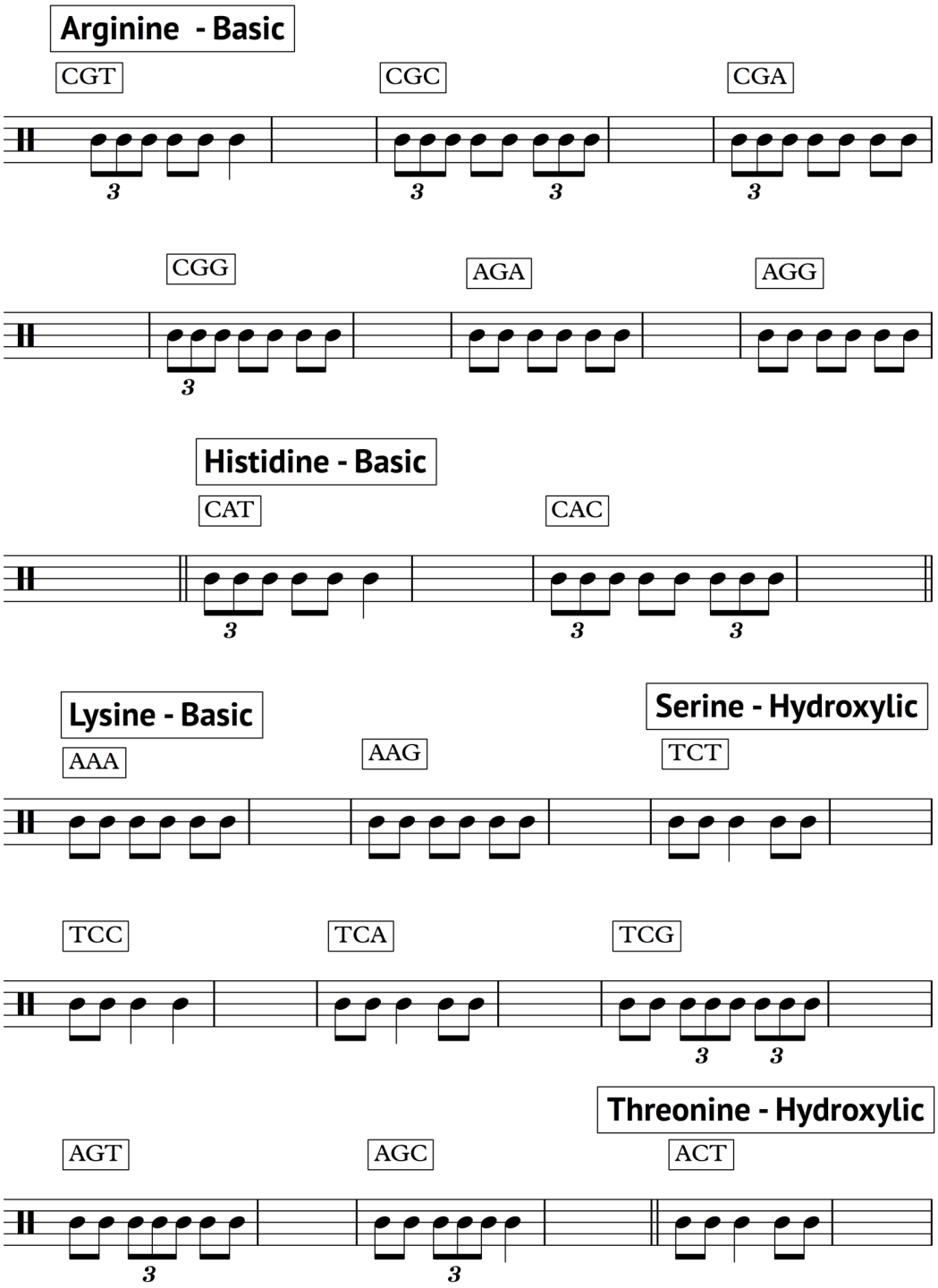

Figure A2.2a: Rhythmic Codons built with the vocabularies in Figure A2.1.

Accepted for publication in journal *Artificial Life* **26(3) -** Pre-publication draft

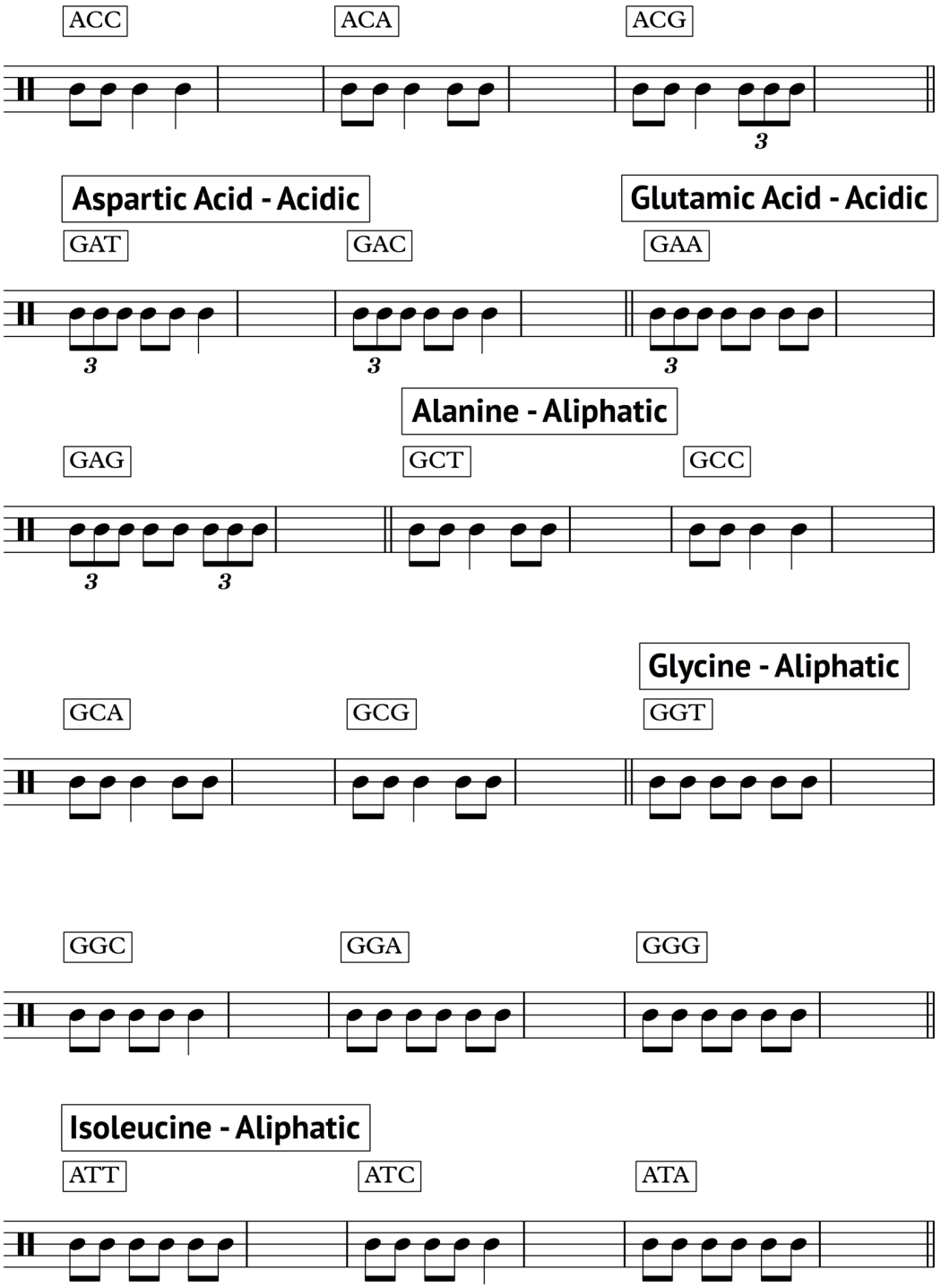

Figure A2.2b: Continuation of Figure A2.2a.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

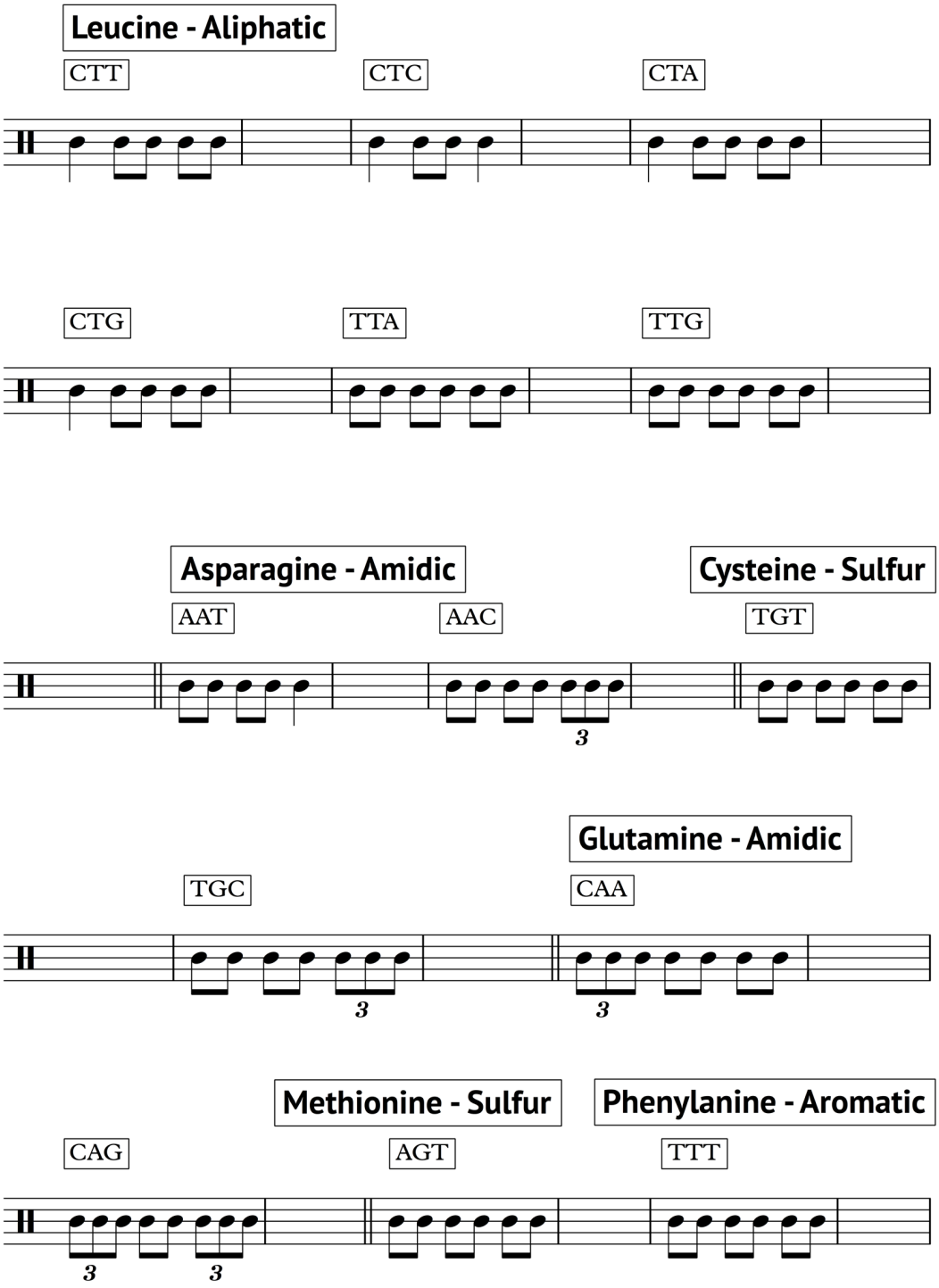

Figure A2.2c: Continuation of Figure A2.2b.

Accepted for publication in journal *Artificial Life* **26(3) -** Pre-publication draft

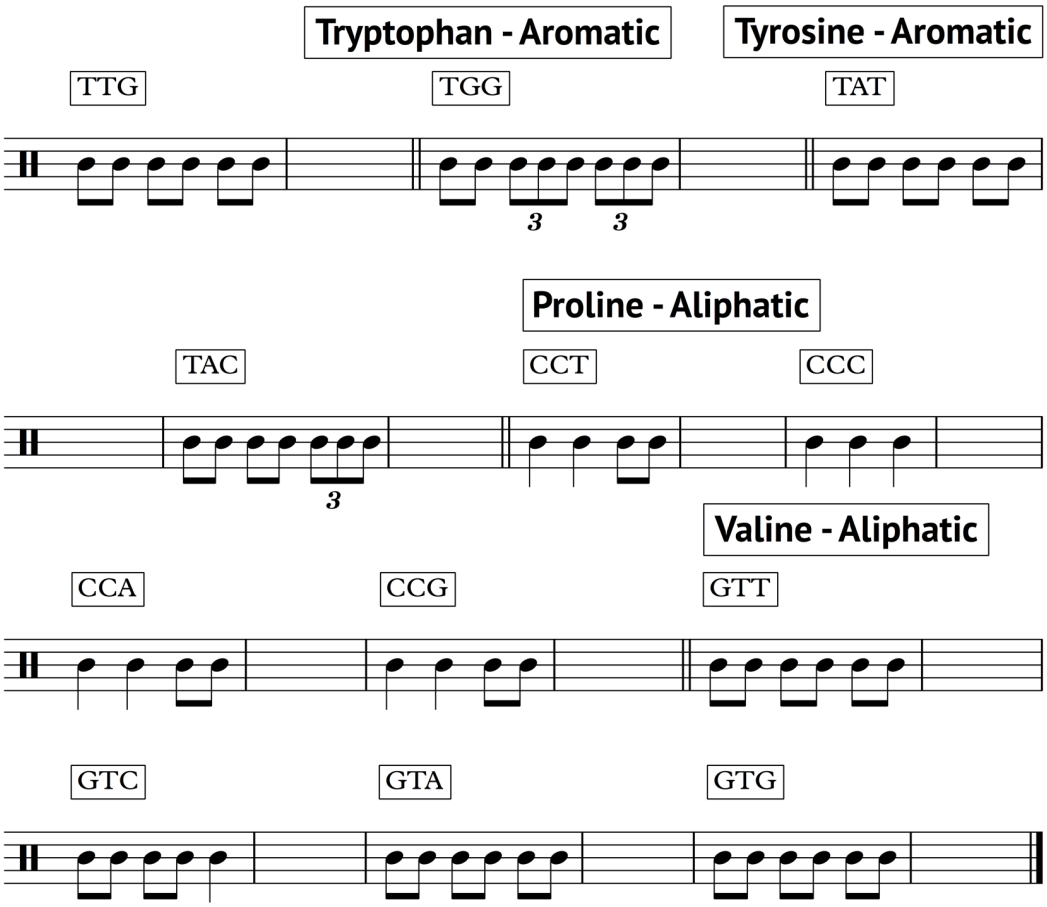

Figure A2.2d: Continuation of Figure A2.2c.

Accepted for publication in journal *Artificial Life* **26(3) -** Pre-publication draft

Appendix 3: Samples of the Score

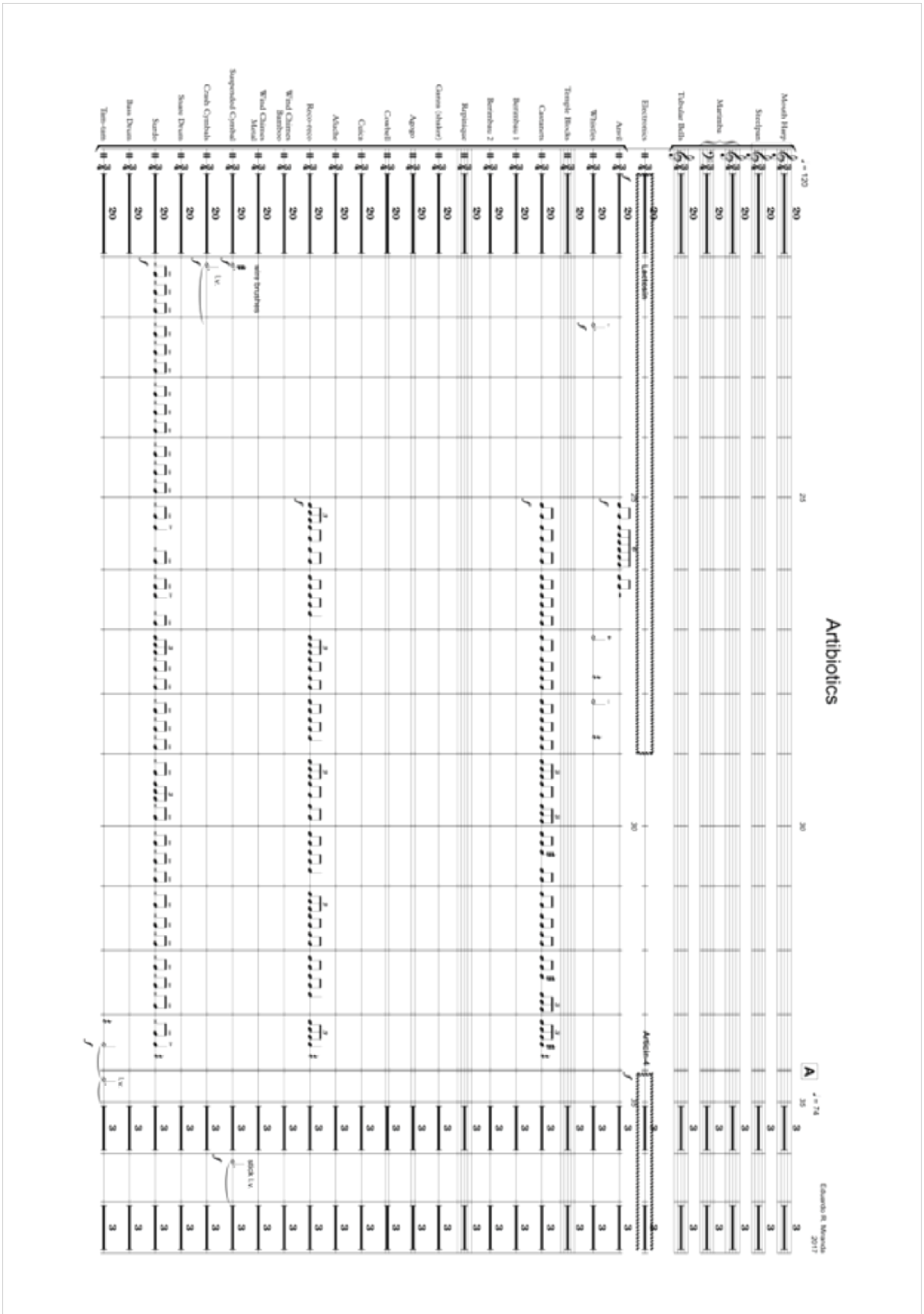

Figure A3.1: Page 1 of *Artibiotics*' score.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

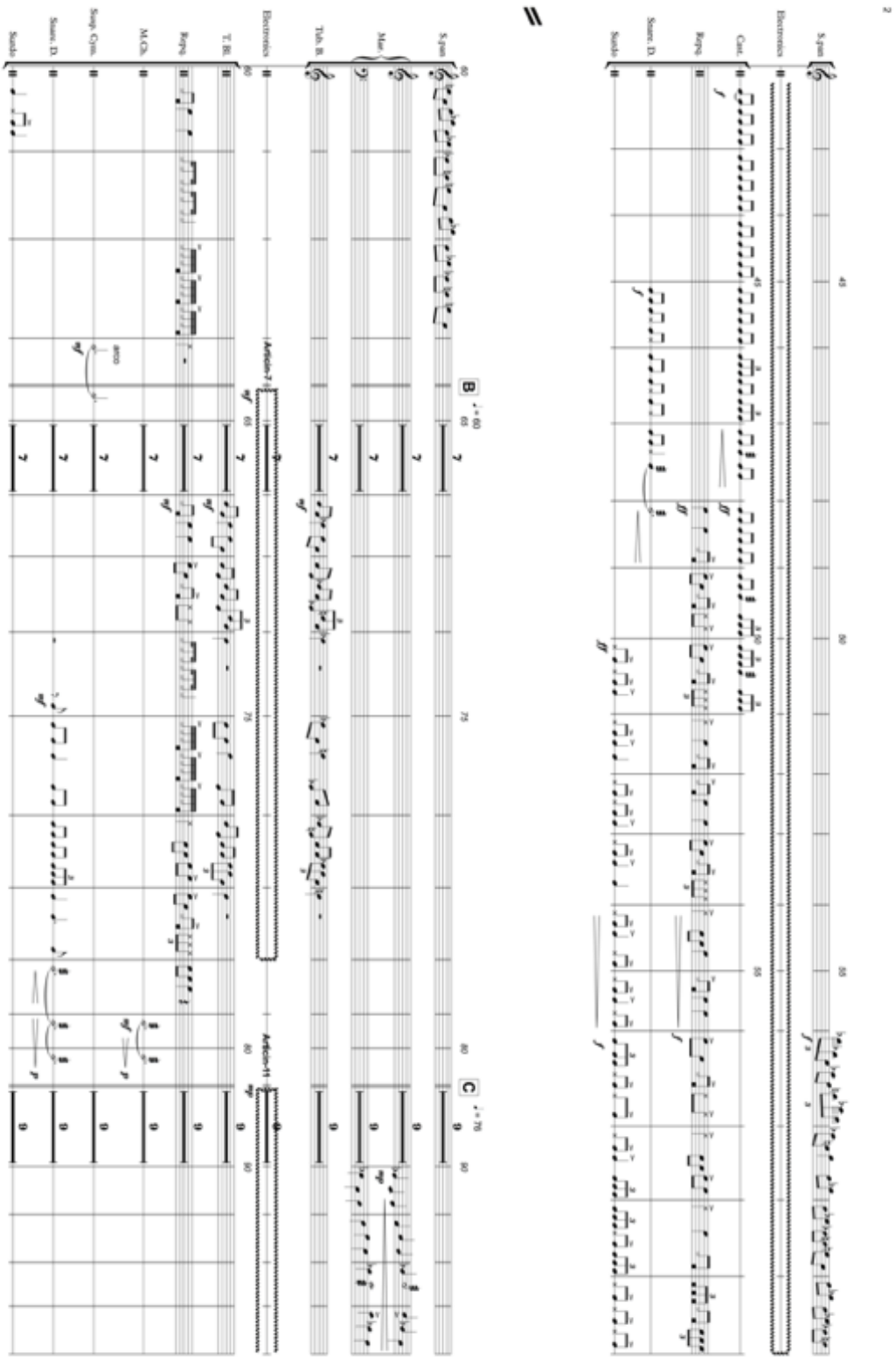

Figure A3.2: Page 2 of *Artibiotics*' score

Accepted for publication in journal *Artificial Life* **26(3) -** Pre-publication draft

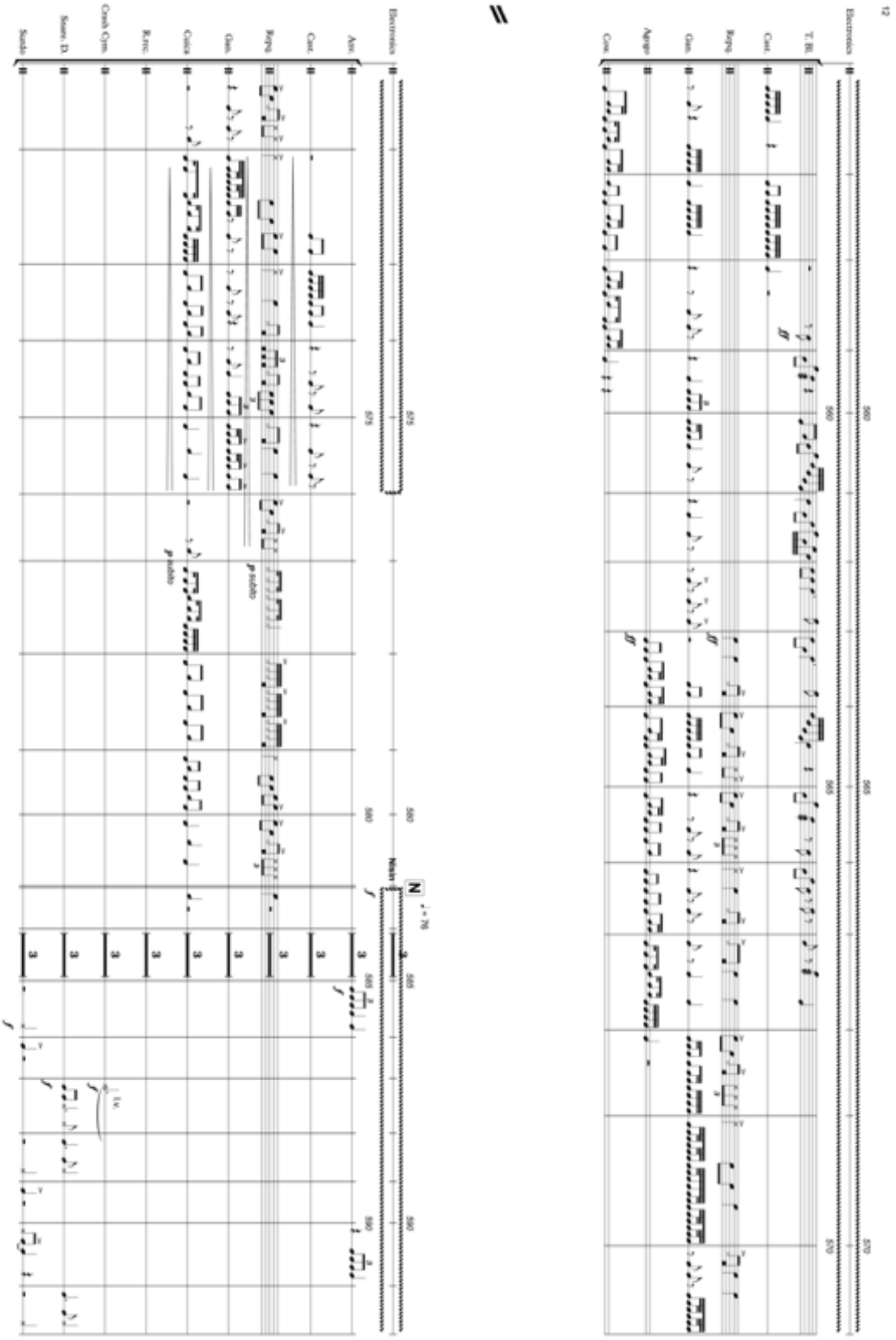

Figure A3.3: Page 12 of *Artibiotics*' score.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

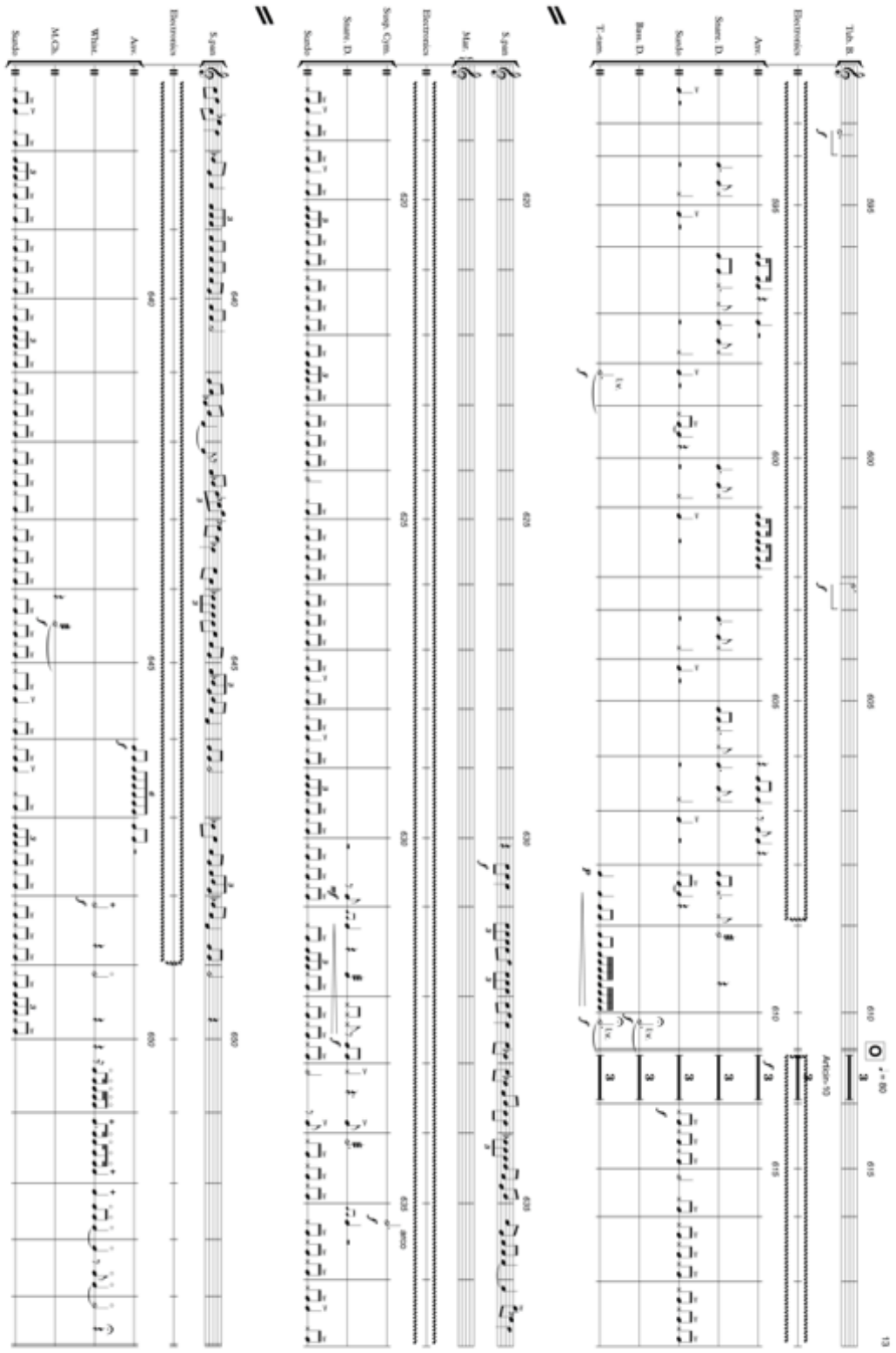

Figure A3.4: Page 13 of *Artibiotics*' score.

Accepted for publication in journal *Artificial Life* 26(3) - Pre-publication draft

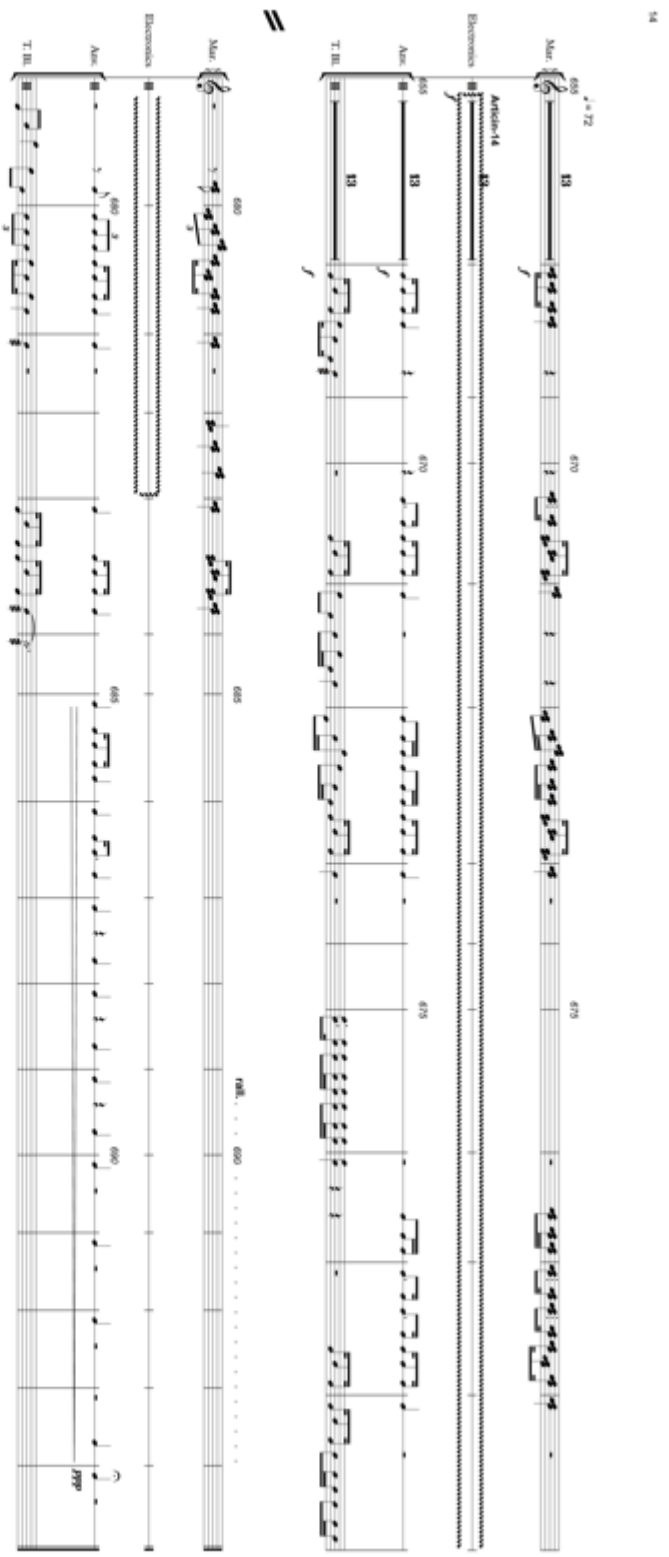

Figure A3.5: Page 14 of *Artibiotics*' score. This is the last page of the score.