UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Programa de Maestría y Doctorado en Música

Facultad de Música
Centro de Ciencias Aplicadas y Desarrollo Tecnológico
Instituto de Investigaciones Antropológicas

AMI: herramienta para la composición algorítmica y calificación de datos simbólicos

TESIS
QUE PARA OPTAR POR EL GRADO DE

DOCTOR EN MÚSICA (TECNOLOGÍA MUSICAL)

PRESENTA
FRANCISCO COLASANTO

TUTOR O TUTORES PRINCIPALES
DR. JORGE RODRIGO SIGAL SEFCHOVICH (ESCUELA NACIONAL DE ESTUDIOS SUPERIORES, MORELIA, UNAM)

MIEMBROS DEL COMITÉ TUTOR
DR. JAVIER ALVAREZ FUENTES (ESAY)
DR. OSCAR PABLO DI LISCIA (UNQ)

MORELIA, MEXICO OCTUBRE 2021

Esas ambigüedades, redundancias y deficiencias recuerdan las que el doctor Franz Kuhn atribuye a cierta enciclopedia china que se titula Emporio celestial de conocimientos benévolos. En sus remotas páginas está escrito que los animales se dividen en (a) pertenecientes al Emperador, (b) embalsamados, (c) amaestrados, (d) lechones, (e) sirenas, (f) fabulosos, (g) perros sueltos, (h) incluidos en esta clasificación, (i) que se agitan como locos, (j) innumerables, (k) dibujados con un pincel finísimo de pelo de camello, (l) etcétera, (m) que acaban de romper el jarrón, (n) que de lejos parecen moscas.

Jorge Luis Borge. *El idioma analítico de John Wilkins* (Borges, 1960)

Agradecimientos

De manera muy especial al Dr. Rodrigo Sigal Sefchovich, tutor principal de este trabajo.
A los Dres. Oscar Pablo Di Liscia y Javier Álvarez Fuentes.
A mis sinodales: Dra. Adina Izarra, Dr. Jorge David García y Dr. Edmar Soria.
A la Dra. Silvana Casal
A todas aquellas personas que me ayudaron con sus invaluables aportes.
A la Facultad de Música de la UNAM.

Jurado para el examen de doctorado:

Dr. Pablo Oscar Di Liscia

Dr. Javier Álvarez Fuentes

Dra. Adina Izarra

Dr. Jorge David García

Dr. Edmar Soria

Tabla de contenidos

Índice de figuras

Índice de videos

Índice de tablas

Introducción general

Como se podrá leer en este trabajo, el uso de herramientas y técnicas para la creación musical con procesos algorítmicos se remonta al siglo X y quizá antes. Por lo tanto el trabajo que aquí se presenta no pretende ser novedoso en ese sentido sino en el modo de implementación. De esta manera, lo novedoso de esta propuesta radica en el hecho de unir un sistema de generación de datos musicales simbólicos a partir de algoritmos, con la posibilidad de clasificar dicho material a partir de diferentes técnicas de análisis y todo ello implementado en un entorno de trabajo amigable y de uso común en el mundo de la creación e interpretación musical como lo es Ableton Live.

Este trabajo presentará una herramienta denominada *AMI* (Algorithmic Music Interface) que brinda la posibilidad de generar, a partir de procesos algorítmicos, diferentes perfiles de datos que serán traducidos en alturas, ritmos y dinámicas que podrán ser almacenados en Ableton Live como clips MIDI. Los tipos de algoritmos utilizados son: gramaticales, caóticos, aleatorios, genéticos y autómatas celulares, y el análisis de los datos almacenados podrá llevarse a cabo a partir de cuatro diferentes técnicas: distancia de edición, n-gramos, entropía, similitud cronotónica. Una vez analizados los clips generados, y luego de almacenar dichos análisis en un base de datos, estos serán relacionados entre ellos a partir de una algoritmo de “vecino cercano”.

Este trabajo esta dividido en cinco capítulos. En el primero se introduce al lector en los antecedentes de la composición algorítmica, y se detalla la metodología del trabajo realizado, como así también su justificación y objetivos. En el segundo se explica, de manera pormenorizada, los rudimentos matemáticos de los diferentes algoritmos utilizados

para la generación de datos simbólicos. En el tercero se muestra cómo se implementaron dichos algoritmos en el entorno de programación Max for Live. En el cuarto se detallan los tipos de análisis que se utilizaron para crear una base de datos que permitirá la clasificación de los clips generados y se explica cómo se implementa dicho análisis y clasificación dentro de *AMI*. Por último, en el capítulo quinto se presentan dos ejemplos musicales creados dentro de este entorno.

Esta tesis puede leerse en su versión ePub (haciendo accesible al lector su contenido multimedia) o en su versión PDF, pudiéndose ver y escuchar los ejemplos de audio y video desde la web https://www.drzoppa.com/ami

Capítulo 1. Composición algorítmica

1.2 - Introducción

Algoritmo, según la Real Academia Española es un: “Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.”[1]. Tomando en consideración esta definición, y aplicándola a la composición musical, se puede decir entonces que, cuando un compositor o compositora se plantea como método de trabajo la generación automática de uno o más parámetros musicales utilizando un conjunto de operaciones dadas, está usando un proceso algorítmico de composición. Otros autores la definen como: “la composición a través de métodos formalizables” (Nierhaus, 2009), “las estrategias compositivas que tienen como punto de partida algún método para la construcción musical” (Hedelin, 2008), “la aplicación de un algoritmo rígido y bien definido al proceso de componer música” (Jacob, 1996).

Utilizar procesos algorítmicos como una manera de asistencia para la generación de parámetros musicales (alturas, ritmos, silencios, articulaciones, dinámicas, etc.) no implica necesariamente que el oyente reconozca dichos procesos en el sentido que pueda identificar qué tipo de algoritmo se utilizó a partir de un determinado resultado sonoro, sino que el compositor o compositora hacen uso de estos recursos como una forma de generar datos de manera tal que puedan ser usados en su obra, con o sin modificación, como materia prima para su posterior utilización de manera más o menos evidente para el oyente. Laurie Spiegel en su artículo *Thoughts on Composing with Algorithms* (Spiegel, 2018) menciona seis maneras en que ha utilizado procesos algorítmicos para la creación musical; estas seis aproximaciones, si bien son específicas de dicha autora, resumen las

1 https://dle.rae.es/algoritmo

diferentes maneras en que se hace uso de esta técnica:

1) Alusión: simular muy aproximadamente un hecho natural, más como una percepción del proceso o de la forma, que una réplica exacta.

2) Análisis inverso: plasmar en la computadora reglas basadas en la música de éxito del pasado.

3) Modelado científico: implementar, utilizando un conjunto de reglas codificadas por software, generadores de datos diseñados para ser cognitivamente significativos o comprensibles.

4) Imitación del proceso: codificar en un programa informático las reglas por las que algún fenómeno natural se produce, se desarrolla o progresa.

5) Mimetización del resultado del proceso: mapear literalmente datos no musicales en variables musicales.

6) Mixta: combinaciones de las anteriores.

Para este fin, existen diferentes técnicas dentro de esta práctica. Diversos autores dividen dichas técnicas en dos o tres grupos de acuerdo al criterio utilizado; así se diferencia, por ejemplo, las composiciones algorítmicas que utilizan las reglas arbitrarias que el compositor decida versus las que utilizan un corpus de reglas tomadas del análisis de obras previas (Mazurowski, 2012), las que utilizan técnicas de la Inteligencia Artificial

versus las que están basadas en modelos de creación humanos (Fernández & Vico, 2013), las que utilizan procesos estocásticos en los cuales los eventos son generados de acuerdo a las características aleatorias de un proceso determinado versus las que utilizan permutaciones de elementos previamente generados (que incluso puede ser material de obras preexistentes) (Dodge & Jerse, 1985), las composiciones basadas en la generación de notas (elementos discretos) de las basadas en la generación de sonido (elementos continuos) (Agon, Assayag, & Bresson, 2006), entre otros (figura 1).

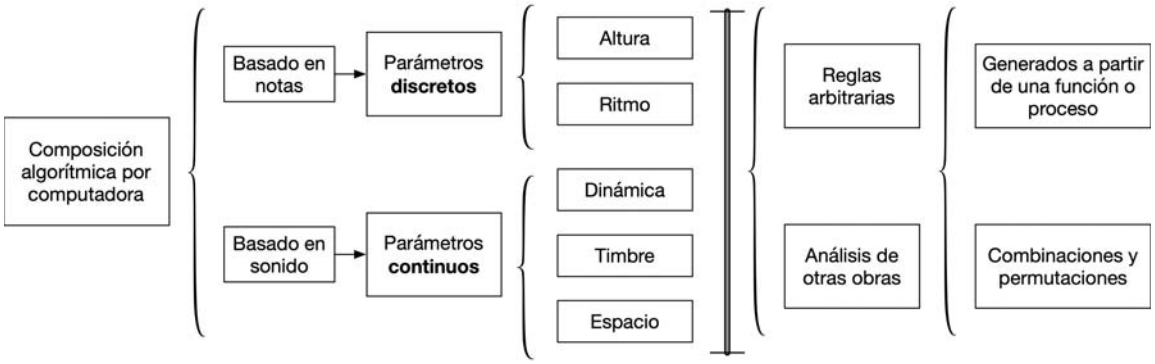


Figura 1: categorías de técnicas

1.2.1 - Antecedentes históricos de la composición algorítmica

Crear una lista detallada de los compositores que a lo largo de la historia emplearon técnicas algorítmicas para la composición musical es una tarea demasiado ambiciosa y excede las necesidades de esta tesis. Sin embargo, a continuación, se presentará una acotada selección de compositores occidentales que permitirá demostrar que dichas técnicas no son exclusivas del arte del siglo XX y XXI.

Uno de los antecedentes históricos más citados (Alsop, 1999; Ariza, 2011; Nierhaus, 2009) es

el de Guido da Arezzo (c. 991 - posterior a 1031). Este monje Benedictino, músico y teórico, contribuyó de manera importante con el desarrollo de la notación musical ya que introdujo la nomenclatura hexacordal (ut - re - mi - fa - sol - la - si) que son las sílabas iniciales del *Himno a San Juan Bautista*[2] donde cada sílaba comienza en dicha nota (Apel, 2003), además desarrolló un sistema para mapear de manera automática los versos del texto con relación a las alturas y frases de la melodía[3].

Muchos otros compositores utilizaron procesos algorítmicos para crear material musical como así también hubo matemáticos que se acercaron a la música. Entre estos últimos se puede nombrar como ejemplo a Marín Mersenne (1588 - 1648), quizá más conocido entre los matemáticos por haber sido quién descubrió los llamados primos de Mersenne. Este científico y teólogo francés publicó en 1636 su *Harmonie Universelle* donde detallaba permutaciones de melodías seriales de 1 a 22 notas (Collins, 2018). Otro religioso (jesuita) que trabajó en este terreno fue Athanasius Kircher (1602 - 1680) quien desarrolló su trabajo musicológico desde 1650 denominado *Musurgia Universalis*. En dicho trabajo presenta un sistema de composición algorítmica que consta de tres categorías de palitos de madera rotulados en los que se graban tanto números como valores rítmicos que permiten generar composiciones contrapuntísticas en estilo simple y florido (Nierhaus, 2009). Mauritius Vogt (1669 - 1730) sugirió un método de composición en el que doblaba unos clavos en varias formas para luego arrojarlos al suelo, así configuraba el ascenso y la caída de las alturas musicales a partir de la forma en que habían caído dichos clavos (Loy, 2011). En el siglo XVIII se popularizó un juego llamado *dado musical* en el cual el

2 https://es.wikipedia.org/wiki/Ut_queant_laxis

3 Muchas veces se le atribuye la creación de un sistema mnemotécnico denominado Mano guidoniana, aunque esta no existe en los textos de Guido da Arezzo. Ver “Guidonian hand” en el Harvard Dictionary of Music (Apel, 2003).

jugador podía crear, con base en las posibilidades combinatorias de cierto material dado, una composición sin necesidad de conocimientos musicales. Para cada compás se podía escoger, a través de un dado, el que lo seguía sin correr el riesgo de crear sucesiones incoherentes de notas (Hed, Gjerdingen, & Levin, 2015; Lin, Liu, Jang, & Wu, 2015; Nierhaus, 2009). Algunos compositores famosos que hicieron uso de algún tipo de juego de dados para componer fueron Carl Philipp Emanuel Bach, Maximilian Stadler, Joseph Haydn y Wolfgang Amadeus Mozart (Hedges, 1978).

Los antecedentes expuestos hasta aquí ilustran el uso de procesos algorítmicos sin la ayuda de computadoras. Para mencionar algunos trabajos donde ya se hace uso de las mismas hay que acercarse más a nuestros días, específicamente a 1957, cuando se estrenó una obra a la que muchos autores consideran como la fundacional en cuanto al uso de computadoras para la composición algorítmica. Esta obra es la *Illiac Suite*, para cuarteto de cuerdas, compuesta por Lejaren Hiller (1924 - 1994) y Leonard M. Isaacson (1925) (Ames, 1987; Anders & Miranda, 2011; Supper, 2001). Esta pieza fue creada utilizando la computadora *ILLIAC-I* (Illinois Automatic Computer) de la Universidad de Illinois. La obra, de cuatro movimientos, se compuso utilizando diferentes técnicas de generación aleatoria de notas, controladas por reglas de contrapunto del siglo dieciséis, paradigmas de la música dodecafónica y cadenas de Markov (Essl, 2007). Sin embargo Charles Ames habla en su artículo *Automated composition in retrospect 1956-1981* acerca de otra obra llamada *Push Button Bertha,* creada por Martin Klein y Douglas Bolitho utilizando una computadora DATATRON, la cual fue trasmitida por radio en 1956 (Ames, 1987), es decir que sería previa a la *Illiac Suite*.

Entre los compositores que hicieron de estas técnicas una herramienta consistente dentro

de sus obras están el griego Iannis Xenakis (1922 - 2001) y el alemán Gottfried Michael Koenig (1926). “Creyendo que el oyente experimenta solo los aspectos estadísticos de la música serial, estos compositores razonaron que un mejor enfoque sería componer directamente usando técnicas probabilísticas en lugar de técnicas seriales” (Loy, 2011). La educación temprana de Xenakis fue principalmente como ingeniero, y cuando se mudó a París en 1947 no solo estudió con los compositores Arthur Honegger (1892 - 1955), Darius Milhaud (1892 - 1974) y Olivier Messiaen (1908 - 1992), sino que también trabajó con el arquitecto Le Corbusier (1887 - 1965) con quien colaboraría en una serie de proyectos importantes entre los que se encuentra el famoso Pabellón Phillips construido para la Exposición Universal de Bruselas de 1958. En 1964 publicó *Musiques formelles: nouveaux principes formels de composition musicale* (Xenakis, 1963) que luego será traducido al inglés como *Formalized Music: Thought and Mathematics in Composition* (Xenakis, 1971). Allí el autor explica sus técnicas de composición utilizando funciones matemáticas estocásticas. La reedición de 1992 contiene, además, ejemplos de código en lenguaje FORTRAN para el software GENDYN (GENeration DYNamique stochastique).

Koenig, por su parte, escribió entre los años 1964 y 1966 su software PROJECT 1. Dicho programa estaba estructurado alrededor de la composición serial. En una entrevista que este compositor le cedió a Curtis Roads, se le preguntó acerca de este software a lo que Koenig respondió (Koenig et al., 1978):

> La idea básica era hacer uso de mi experiencia en la música serial y también las consecuencias aleatorias extraídas de la música en serie. Traté de crear un modelo en el que se describieran ciertas decisiones básicas, y donde, además de eso, el usuario, el compositor, tuviera alguna influencia sobre las variables musicales. De esa manera, se podrían pedir cualquier cantidad de variantes que evolucionaran del mismo principio básico;

> se podrían comparar y ver hasta qué punto las características musicales establecidas en el programa eran realmente experimentadas en la música.

Entre los compositores latinoamericanos pioneros en esta disciplina se puede mencionar a Julio Estrada (1943) y Horacio Vaggione (1943). Estrada nació en la Ciudad de México y, gracias a una beca del gobierno francés, estudió en París entre 1965 y 1969. Allí cursó armonía con Nadia Boulanger (1887 - 1979), análisis musical y composición con Messiaen y tomó contacto con el trabajo de Xenakis (luego trabajaría en el CEMAMu[4] donde compuso su obra Eua'on). En 1973 estableció el seminario de composición en la Universidad Nacional Autónoma de México (McHard, 2008). En 1984 publicó su libro *Música y teoría de grupos finitos (3. Variables booleanas)* (Estrada & Gil, 1984) donde expuso parte de sus técnicas de composición algorítmica.

Horacio Vaggione nació en la ciudad de Córdoba, Argentina. Estudió en la Universidad Nacional de Córdoba y con el compositor Juan Carlos Paz (1897 - 1972), luego con Lejaren Hiller en la Universidad de Illinois (quien lo introdujo en la composición por computadoras) y en la Universidad Paris VIII (Budón, 2000). Su trabajo con procesos algorítmicos se remonta a 1970. De ese año es su artículo *Obtención de formas musicales a través de la codificación digital de textos* (Vaggione, 1970). Allí el autor dice:

> Esta es una investigación sobre las posibilidades de generación automática de música por medio de fichas perforadas. Tal investigación tiene relación con la curiosidad de ver qué clases de música pueden estar contenidas en una forma lógica "venida de fuera", independiente de la memoria y de la voluntad del compositor.

La lista de compositores que trabajaron y trabajan actualmente con técnicas algorítmicas

---

4 https://en.wikipedia.org/wiki/CCMIX

para la composición musical es vasta. Para mayor detalle se recomienda leer el artículo *AI Methods in Algorithmic Composition: A Comprehensive Survey* (Fernández & Vico, 2013) donde los autores detallan una lista de creadores, obras y técnicas de composición utilizadas que abarcan varias décadas.

1.2.2 - Antecedentes de la musicología por computadora

El campo de la Recuperación de Datos Musicales (MIR por su sigla en inglés) busca acceder a datos contenidos tanto en representaciones simbólicas de la música (MIDI, MusicXML) como en archivos de audio (Chen et al., 2019). La representación simbólica de una obra musical (alturas, duraciones, dinámicas, articulaciones, etc.), entonces, puede convertirse en números que permiten implementar búsquedas estadísticas de manera sencilla (Brown, 1999). Esto posibilita el desarrollo del análisis musical a través de computadoras, una práctica que tiene sus inicios en la década de 1960 (Mor, Garhwal, & Kumar, 2019) y que dio lugar a una disciplina denominada Musicología por Computadora. En 1966, utilizando computadoras extremadamente caras y complejas (no tenían una interfaz visual como un monitor, sino que utilizaban tarjetas perforadas para ingresar datos), se realizaron dos proyectos en Estados Unidos que fueron los precursores de esta disciplina, ellos son el Digital Alternate Representation of Musical Scores (DARMS), que fue desarrollado entre 1966 y 1976 en la Universidad de Columbia y el Intermediary Musical Language (IML) y su versión mejorada Fastcode[5] desarrollado en la Universidad de Princeton[6] (Hewlett & Selfridge-Field, 1991). Sin embargo, dichas herramientas

5 Se puede descargar el manual de Fastcode desde https://wiki.ccarh.org/images/5/53/22_Fastcode.pdf

6 Raymond F. Erickson menciona una sesión tormentosa en la American Musicological Society en Santa Bárbara (1967) en la que se suscitaron acaloradas discusiones entre los académicos que ponderaban los métodos tradicionales aplicados a la musicología histórica y aquellos que habían adoptado a las computadoras como su principal herramienta para la investigación (Erickson, 1968).

informáticas eran muy complejas de utilizar por los no programadores. Es en la década de 1970, a partir de lenguajes de programación como *COBOL*, *SNOBOL*, y *Pascal*, que se desarrollaron otras plataformas más completas. Así se introdujeron entornos de trabajo como el lenguaje de entrada de símbolos musicales llamado MUSTRAN, que permitía analizar datos, imprimirlos (a través de SMUT) y obtener un resultado sonoro a partir de la síntesis de sonido utilizando *MUISIC V* [7] (Byrd, 1977). Otros ejemplos mencionados por Donald Byrd (1946) son el sistema *Teletau* creado en el Conservatorio de Florencia (1973) y *Musikus* en la Universidad de Oslo (1976). Es de destacar que las investigaciones realizadas por Allen Forte (1926 - 2014) sobre la estructura de la música atonal ya incluían sofisticadas aplicaciones informáticas de análisis y, según Bo H. Alphonce, "(…) todo el aparato fundamental de términos, conceptos, relaciones y reglas de inferencia pertenecientes a la teoría de los complejos de conjuntos se desarrolló en estrecha interacción con una serie de programas informáticos" (Alphonce, 1980).

La década de 1980 trajo dos avances muy importantes que impactarían en esta disciplina: la introducción de las computadoras personales[8] (Selfridge-Field, 1996) y la creación del protocolo de comunicación MIDI[9]. Esto permitió el desarrollo de software comerciales llamados Digital Audio Workstation (DAW) que comenzaron a aglutinar diferentes aplicaciones dentro de una sola plataforma (grabación de audio, secuencias MIDI, impresión de partituras, etc.). A partir de allí los compositores e intérpretes pudieron grabar y editar música de manera rápida y eficiente, abriendo las posibilidades para crear

---

7 MUSIC IV fue desarrollado en los Laboratorios Bell en 1963 por M. Mathews y J. Miller utilizando una computadora IBM 7094 (Curtis Roads & Strawn, 1996).

8 https://es.wikipedia.org/wiki/Computadora_personal

9 https://www.midi.org/articles/midi-history-chapter-6-midi-is-born-1980-1983

complejas obras musicales (Pople, 1992). Algunos ejemplos de estos software son: *Steimberg Pro-16*, *SoundDesigner*, el *Opcode MIDIMAC*, el *C-Lab notator*, entre muchos otros. También aparecen los primeros software de uso académico que corrían en computadoras personales. Entre ellos es posible nombrar: *Cmix* (Paul Lansky en la Universidad de Princeton. 1984), *Csound*[10] (Barry Vercoe & R. Karstens en el MIT. 1986), *Max*[11] (Miller Puckette en el IRCAM. 1988) y *Humdrum*[12] (David Huron 1980), entre otros. Este último fue (todavía lo es) de gran importancia para la musicología por computadoras ya que se trata de un software creado específicamente para tal fin. Todo esto sumado a la posibilidad de contar con bases de datos relacionables (como el proyecto *RISM*[13]) ayudó al desarrollo esa disciplina.

Otra consecuencia de la recuperación de datos musicales simbólicos es la posibilidad de ser aplicados a procesos de composición algorítmica. Un ejemplo de esto es el trabajo de David Cope (1941) y su software EMI (Cope, 1992) o las ideas de Byrd acerca del desarrollo de un software que integra el análisis musical, la creación, la impresión de partituras, etc. (Byrd, 1977).

1.2.3 - Primeros programas de computadora para la CAC

La primera librería creada para la Composición Asistida por Computadora (CAC) es *MUSICOMP*. Fue desarrollada por Robert Baker (1933 - 2000) y Lejaren Hiller en la Universidad de Illinois y fue con esta herramienta que se compuso la *Illiac Suite*.

---

10 https://csound.com/

11 https://cycling74.com/

12 https://www.humdrum.org/

13 http://www.rism.info/en/home.html

La biblioteca *MUSICOMP* (una serie de sub-rutinas en lenguaje *FORTRAN*) incluía herramientas para seleccionar elementos de una lista de acuerdo con una distribución de probabilidad; podía “barajar” elementos aleatoriamente en una lista, manipular listas de alturas, imponer reglas melódicas y coordinar las líneas rítmicas (Gérard Assayag, 1998; Curtis Roads, 2015).

Como ya se ha mencionado, otro precursor fue Gottfried Michael Koenig con su software *PROJECT 1* el cual fue “Concebido por primera vez en el Instituto de Matemáticas de la Universidad de Bonn durante una visita en 1963. En ausencia de un control informático directo del equipo de estudio en esta etapa, se utilizaron cintas de control analógico en lugar del secuenciador, codificándose los voltajes como funciones de frecuencia mediante un par de moduladores y demoduladores” (Manning, 2004). En la página web[14] oficial de Koenig es posible leer la siguiente descripción de su software:

> El programa informático Project 1 (PR1) nació en 1964 del deseo de probar las reglas de composición de la música serial, reglas que fueron objeto de una animada discusión en esos días. Sin embargo, pronto se hizo evidente que las reglas basadas en listas de parámetros y permutaciones de filas no pueden describirse sin planes concretos para una composición; en cualquier caso, la combinación sistemática de todos los puntos de partida concebibles habría dado lugar a una cantidad incalculable de resultados que no podrían haberse evaluado sin un plan concreto para una composición. Por lo tanto, era necesario limitar el procedimiento a un modelo de composición que contenga elementos importantes del método en serie, y probar ese modelo en varias condiciones con diferentes objetivos musicales en mente (…). El modelo en el que se basa el programa procede de un par de opuestos, "regular / irregular" (el principio RI), inspirado en la no repetición de elementos seriales ("irregular") y en filas de multiplicación formadoras de grupos. Entre estos extremos hay cinco etapas intermedias, de modo que el compositor puede elegir entre un total de siete "procesos". En PR1, el principio RI se aplica al instrumento de parámetros, retardo de entrada, tono, registro de octava y dinámica. A cada parámetro se le asigna una lista en la

14 http://koenigproject.nl/project-1/

> que el compositor ingresa los valores de los parámetros deseados. Durante el proceso de composición se producen "secciones", en cada una de las cuales el compositor puede determinar un "proceso" para cada parámetro (…).

A partir de mediados de la década de 1980 comienzan a aparecer una serie de software dedicados a la composición algorítmica. Uno de los factores que contribuyeron especialmente a esto fue la aparición del protocolo MIDI (Musical Instrument Digital Interface)[15] que permitió la comunicación entre máquinas y controladores de manera sencilla, y al desarrollo de nuevos lenguajes de programación (Gérard Assayag, 1998). Uno de estos programas fue *Common Music*, creado por Rick Taube (1953) en 1989[16]. Este software esta basado en el lenguaje *Common LISP* desarrollado en 1956 por investigadores del campo de la inteligencia artificial. La música, una forma de arte basada en el tiempo, a menudo se concibe como una sucesión de eventos, es por ello que el tipo de arquitectura de LISP es idóneo para la composición algorítmica (Simoni, 2003). En la web de dicho software[17] se lee la siguiente descripción: “Common Music es un sistema de composición musical que transforma representaciones algorítmicas de alto nivel de procesos y estructuras musicales en una variedad de protocolos de control para la síntesis y visualización de sonido”.

Los desarrollos realizados en este mismo período en el IRCAM (Institut de Recherche et de Coordination Acoustique Musique)[18] relacionados con la Composición Asistida por Computadoras son especialmente destacables. Todavía, al momento de escribir esta tesis,

---

15 https://www.midi.org

16 http://commonmusic.sourceforge.net/#history

17 http://commonmusic.sourceforge.net

18 https://www.ircam.fr

sigue siendo parte importante de dicha institución y tal es así que posee un equipo de investigación denominado *Représentations musicales*[19] que lleva adelante diversos proyectos en ese campo. Gérard Assayag (1960), director de dicho equipo escribe (Gérard Assayag, 1998):

> La contribución del IRCAM (Institut de Recherche et de Coordination Acoustique Musique) al desarrollo de la Composición Asistida por Computadoras (CAC) merece un tratamiento especial porque es uno de los pocos lugares que ha estado implicado institucionalmente y con perseverancia desde hace aproximadamente quince años en la defensa de estas ideas. Desde principios de los años ochenta, varios equipos sucesivos trabajaron en la dificultad de representar y manejar las estructuras musicales y el conocimiento en una perspectiva compositiva.

En el citado artículo se puede leer también acerca de *Formes*. Este fue el primer entorno CAC salido del IRCAM. Los programadores de dicho software explican que el mismo fue desarrollado en 1981 buscando proveer, a los compositores que realizaban residencias de creación en el IRCAM, de una herramienta poderosa para la investigación y producción musical (incluso para aquellos con poca o ninguna experiencia en el uso de computadoras) (Cointe & Rodet, 1984). *Formes* fue programado en *VLisp*[20] y consistía de una librería de modelos predefinidos de herramientas creadas en *LISP*. En dicho entorno, la tarea del compositor era la de modificar dichos modelos para derivar de allí otros nuevos. En 1987 Lee Boynton y Jacques Duthen lanzaron *PreForm* que era una interfaz gráfica para *Formes* que utilizaba MIDI y que corría en computadoras Macintosh, el cual fue utilizado en muchas producciones del IRCAM, principalmente para controlar dispositivos MIDI en vivo (Eckel, 1988).

---

19 https://www.ircam.fr/recherche/equipes-recherche/repmus/

20 Era un intérprete de LISP y precursor de Le_Lisp desarrollado en Francia. (Steele & Gabriel, 1996)

Entre 1989 y 1990 Francis Courtot (1960) crea *CARLA* (Composition Assistee par Representation Logique et Apprentissage), un software que permitía utilizar una interfaz gráfica para implementar ciertas programaciones lógicas (Gérard Assayag, 1998). Este entorno se proponía “representar el conocimiento musical utilizado por un compositor cuando usa una herramienta de composición asistida por computadora.” (Courtot, 1992). Con *CARLA*, Courtot buscaba resolver al menos tres problemas que el autor encontraba a la hora de crear un sistema de CAC: que el software permitiera crear desde la formalización musical con la que cada compositor se identificara, que tuviera en consideración la falta de conocimiento técnico en lenguajes de programación de la mayoría de los compositores (utilizando una interfaz gráfica mas amigable) y, por último, que se pudiera definir una representación abstracta para dar cuenta formalmente de la aspectos musicales del proceso de composición (Courtot, 1992).

En 1993, el IRCAM lanza *Patchwork* (figura 2), programado en *Common Lisp* (Macintosh), este entorno proporcionaba una interfaz gráfica para el lenguaje *LISP*. Cualquier función se traducía en una caja manipulable gráficamente, donde la sintaxis de programación se definía por las conexiones entre las cajas (Gérard Assayag, Rueda, Laurson, Agon, & Delerue, 1999).

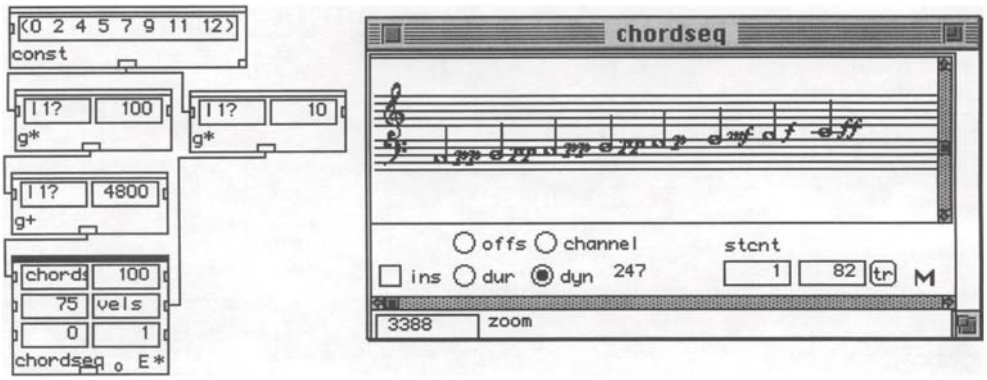


Figura 2: interfaz de *Patchwork* (Assayag et al., 1999)

En 1981 Pierre Boulez (1925 - 2016) estrenó su pieza *Répons* para ensamble de veinticuatro instrumentos, seis solistas, y procesos electrónicos en tiempo real. El equipo utilizado para este fin fue la 4X que era una computadora que había sido creada por Giuseppe di Giugno (1937) en el IRCAM (Gerzso, 1984). En 1985 Miller Puckette (1959), también en el IRCAM, comenzó a programar un software para la 4X y, cuando trabajaba en esto junto con el compositor Philippe Manoury (1952) para la realización de lo que sería su obra *Jupiter*, encontraron que había algunos inconvenientes respecto a cómo el sistema lidiaba con el tiempo. Puckette encontró una solución en 1986 (Puckette, 1986) y superó el problema del tiempo programando un sistema de tiempo real, al que llamó *Max*, en honor al pionero de la música por computadoras Max Matthews (1926 - 2011) ya que "No sólo (Matthews) me influyó enormemente en los pocos meses que trabajamos juntos, sino también su programa *RTSKED* introdujo el enfoque de programación en tiempo real que *Max* adopta" (Puckette, 1991). Para hacer el software más accesible, Puckette, mudó la 4X y su sistema a un Macintosh con soporte MIDI. Luego diseñó una interfaz gráfica (figura 3), a la que llamó Patcher (Lippe, 1991).

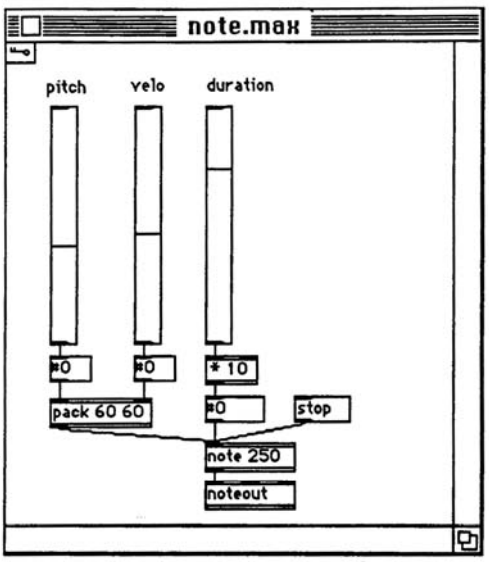


Figura 3: interfaz gráfica de Patcher (Miller Puckette, 1988)

1.2.4 - Estado actual de la composición algorítmica por computadora

Las compositoras y compositores que utilizan de una u otra manera las técnicas de composición algorítmica se renuevan constantemente. Ya no es una disciplina exclusiva de la música académica. Prueba de esto es que algunos software comerciales como *LogicPro*[21], *Ableton Live*[22] y *Reason*[23] incorporan herramientas algorítmicas para la creación sonora. Esto da como resultado la aparición de nuevas músicas de estilos variados que utilizan estas técnicas. Alex McLean (1975) y Nick Collins (1975), por ejemplo, fundaron la agrupación *Algorave*[24] que realiza eventos de música dance creada algorítmicamente (Knotts & Collins, 2018; Savery, 2018). “En las últimas décadas, las comunidades de música algorítmica se han formado alrededor de un número continuo de enfoques, que podemos organizar según la relación entre lo humano y el algoritmo.” (McLean & Dean, 2018a). Otro ejemplo de su creciente utilización es el uso de estas técnicas para la composición de música para videojuegos (Duarte, 2020; Fridenfalk, 2015).

Otro indicador de la relativa expansión de la música algorítmica es que en la plataforma Facebook se pueden encontrar grupos públicos que congregan a los entusiastas de esta disciplina, uno de ellos llamado Algorithmic Composition[25] cuenta, al momento de escribir este texto[26], con 7400 miembros. Por lo antedicho es posible afirmar que existe una tendencia novedosa donde las prácticas antes solo reservadas al mundo académico,

---

21 https://www.apple.com/logic-pro/

22 https://www.ableton.com

23 https://www.reasonstudios.com

24 https://en.wikipedia.org/wiki/Algorave también en https://algorave.com

25 https://www.facebook.com/groups/AlgoComp

26 Martes 30 de marzo de 2021

encuentran cada vez mayor aceptación dentro de ámbitos no académicos (quizá sea un poco arriesgado llamarlos ámbitos populares). En el libro *The Rest Is Noise: Listening to the Twentieth Century* se lee: “Un destino posible para la música del siglo XXI es una "gran fusión" final: artistas pop inteligentes y compositores extrovertidos[27] que hablan más o menos el mismo idioma” (Ross, 2007).

En la esfera de lo académico, la CAC ha experimentado un auge en los últimos 20 años y los software especializados se han multiplicado. Esto se debe, en cierta medida, al poder que las computadoras personales han adquirido en los últimos años y, quizá a consecuencia de ello, a un nuevo interés por parte de las nuevas generaciones de compositores. Miller Puckette en el prefacio del libro *The OM Composer’s Book: Volume One* (Agon et al., 2006), afirma que la Música Generada por Computadoras (CGM por sus siglas en inglés) y la Composición Asistida por Computadora (CAC) tomaron, en un principio, caminos separados. La primera, afianzada mayormente en los ambientes académicos de Estados Unidos parecía ser la respuesta más obvia a las necesidades creativas de los compositores, mientras que la segunda, más desarrollada en Europa, parecía mas una práctica científica que creativa (por lo menos así lo pensaba Miller Puckette). Sin embargo, en el año 2006 cambió su parecer y comenzó a considerar a la CAC como una práctica más compleja y abarcadora. En el mencionado prefacio afirma: “El campo de la CAC en general se está alejando de las construcciones matemáticas e informáticas, y está yendo hacia una relación de trabajo más útil y poderosa con el resto del proceso de composición”. Para apuntalar la afirmación de que las prácticas que involucran a la CAC gozan de muy buena salud, se mencionan, a continuación, algunos

27 Refiriéndose a artistas pop y avant-garde o académicos. En este caso Ross habla de la música de Björk y Osvaldo Golijov.

artículos aparecidos en los últimos tres años que ilustran su uso de una u otra forma. El orden de aparición no está relacionado con su orden de importancia o con su cronología. ·El artículo *Next-generation Computer-aided Composition Environment: A New Implementation of OpenMusic* (Bresson, Bouche, Carpentier, Schwarz, & Garcia, 2017) presenta OpenMusic 7, sucesor de OM 6 que, aunque su estado de desarrollo al momento de escribir esta tesis se encuentra todavía en una fase beta, pretende incorporar mejoras funcionales y gráficas a dicho software. En *A Transformational Modified Markov Process for Chord-Based Algorithmic Composition (Amram, Fisher, Gul, & Vishne, 2020)* los autores presentan un sistema de composición de acordes basado en Cadenas de Markov modificadas "(…) para equilibrar entre la armonía, la familiaridad, y la entropía (…)". En *Algorithmic Music Composition Based on Artificial Intelligence: A Survey* (Lopez-Rincon, Starostenko, & Ayala-San Martín, 2018) los autores realizan un examen de las obras publicadas después de 2010 y que utilizan Inteligencia Artificial para su creación. En febrero de 2018, Oxford University Press publicó *The Oxford Handbook of Algorithmic Music* (McLean & Dean, 2018b), libro que contiene 34 artículos que se ocupan de este tema.

1.2.5 - Software utilizado para CAC

A continuación, se analizarán algunos de los software utilizados en la CAC existentes al momento de escribir esta tesis. Aquí se dividen en dos categorías: aquellos destinados específicamente para dicho propósito y aquellos que incorporan herramientas destinadas a la composición algorítmica dentro de un conjunto mayor de funcionalidades. Entre los primeros se analizarán *OpenMusic*, *OpusModus*, *Nodal* y *Tidal Cycles*; entre los segundos *Max* y *SuperCollider*.

OpenMusic

OpenMusic es un software desarrollado por el equipo de investigadores del IRCAM denominado *Représentations musicales* y, como se describe en su web oficial[28], este equipo trabaja con “las estructuras formales de la música y ambientes creativos para la composición y la interacción musical. Este trabajo encuentra su aplicación en la composición asistida por computadora, performance, improvisación y musicología computacional”. Este software, como ya se mencionó, es un sucesor de *PatchWork* y fue presentado por sus creadores en la *International Computer Music Conference*, que se realizó en Thessaloniki, Grecia, en septiembre de 1997 (Gerard Assayag, Agon, Fineberg, & Hanappe, 1997). Está basado en *LISP* y consta de una interfaz gráfica (figura 4) desde donde es posible interconectar objetos que representan funciones y estructuras de datos[29].

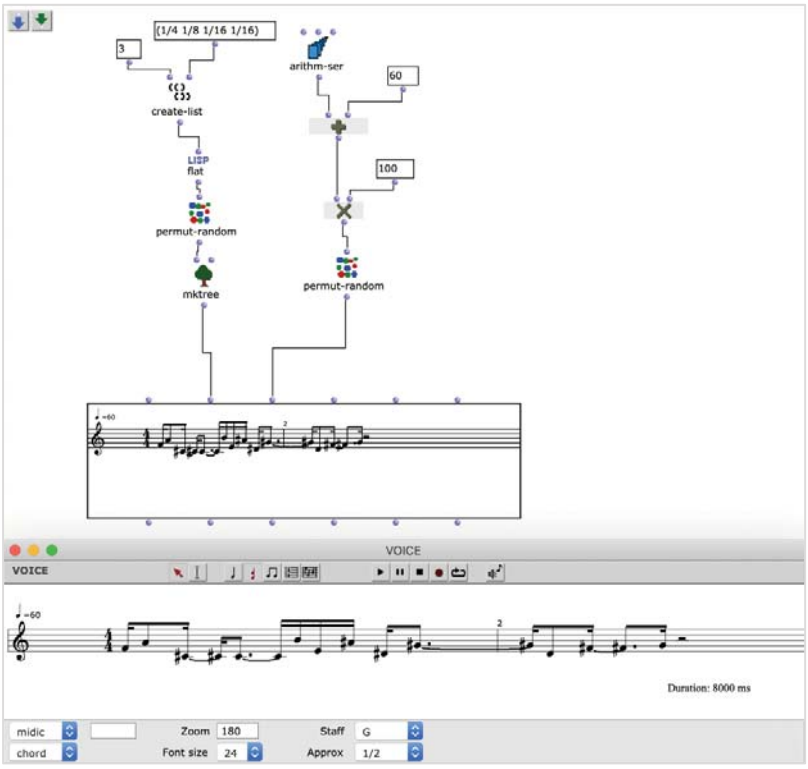


Figura 4: interfaz de OpenMusic

28 https://www.ircam.fr/recherche/equipes-recherche/repmus/

29 https://openmusic-project.github.io/openmusic/

Este software posee una serie de objetos que permiten visualizar, a modo de partitura, el resultado de la manipulación de información que se realiza a partir de los objetos interconectados en el patch. Estos objetos que permiten el uso de una partitura se denominan: *note*, *chord*, *chord-seq*, *multi-seq*, *voice* y *poly*. Cada uno de ellos representa niveles jerárquicos diferentes (figura 5).

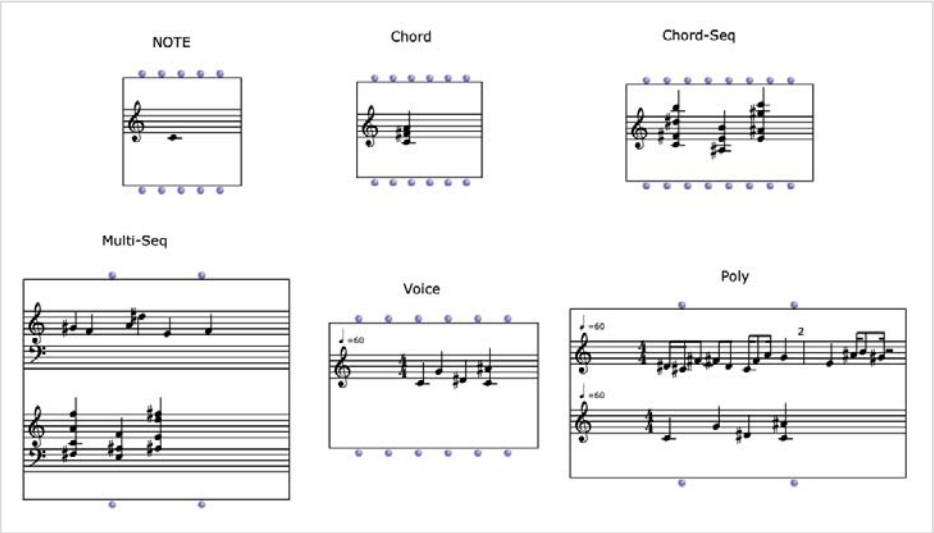


Figura 5: objetos para representar partituras.

OpenMusic posee una serie de objetos divididos en clases y funciones. Las primeras, como en toda programación orientada a objetos, posee sus métodos y atributos. Se puede simplificar diciendo que las clases incluyen a los objetos desde donde se ingresan datos a las funciones y donde se grafican los resultados arrojados por dichas funciones, mientras que las funciones en sí mismas son aquellas que realizan una tarea específica, como, por ejemplo, sumar dos enteros (figura 6).

Este software incluye, al instalarlo, unas sesenta clases diferentes: objetos para recibir y enviar información MIDI, enviar y recibir mensajes dentro del protocolo *Open Sound*

Control[30], crear gráficos cartesianos, generar partituras, y muchos otros; mientras que las funciones incluyen más de doscientos cincuenta objetos que realizan muy variadas tareas matemáticas, lógicas, de combinatoria, etc.

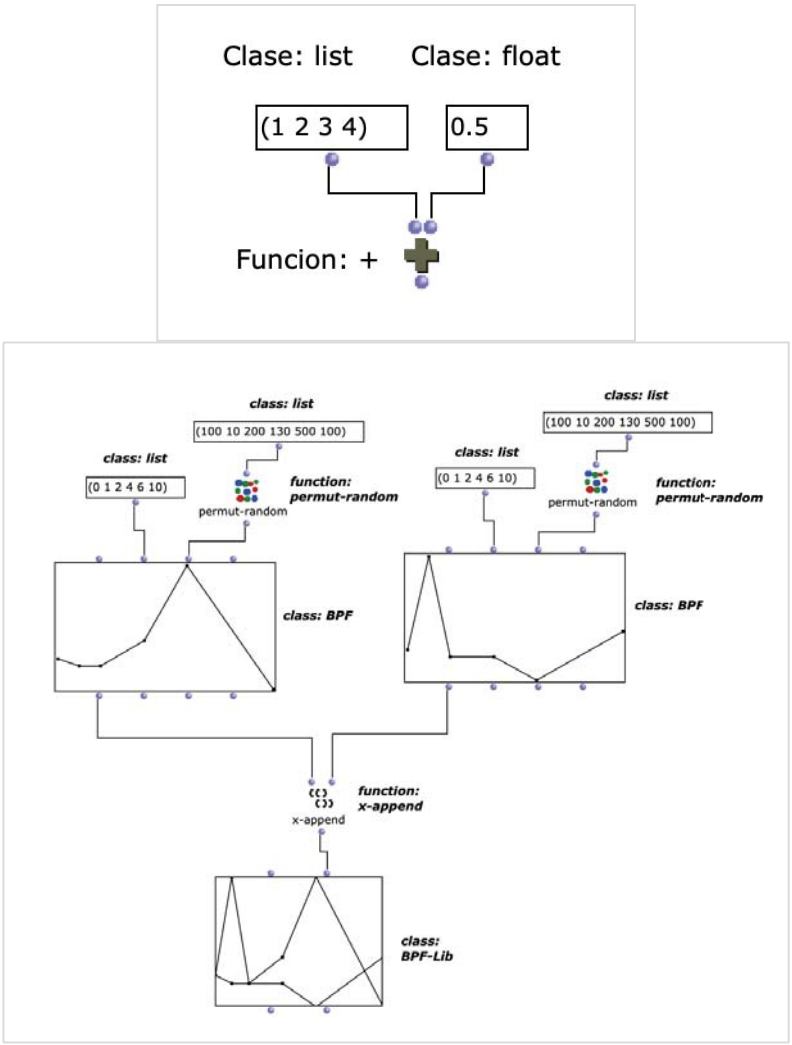


Figura 6: clases y funciones en *OM*

Es posible incorporar librerías externas que aumentan la cantidad de clases y funciones disponibles[31]. Dentro de estas se encuentran librerías tan disímiles como *Morphology* de Jacopo Baboni Schilingi y Frédéric Voisin (Baboni-Schilingi & Voisin, 1999) que “permite el

30 https://www.cnmat.berkeley.edu/opensoundcontrol

31 https://openmusic-project.github.io/libraries

análisis, la clasificación, el reconocimiento de formas y la reconstitución de perfiles y de secuencias tanto numéricas como simbólicas", o la librería *OMMatrix* de Pablo Cetta (Cetta, 2018) "destinada a la generación y transformación de matrices combinatorias y a la realización de operaciones con conjuntos de grados cromáticos", es posible incorporar alrededor de otras cincuenta librerías diferentes más.

Otra particularidad de *OpenMusic* es que cuenta con una interfaz llamada maqueta (figura 7) que permite interconectar y crear cadenas de patches. Esto posibilita, por ejemplo, concatenar secuencias de estructuras musicales creadas cada una en un patch diferente, pudiéndose así estructurar temporalmente el trabajo.

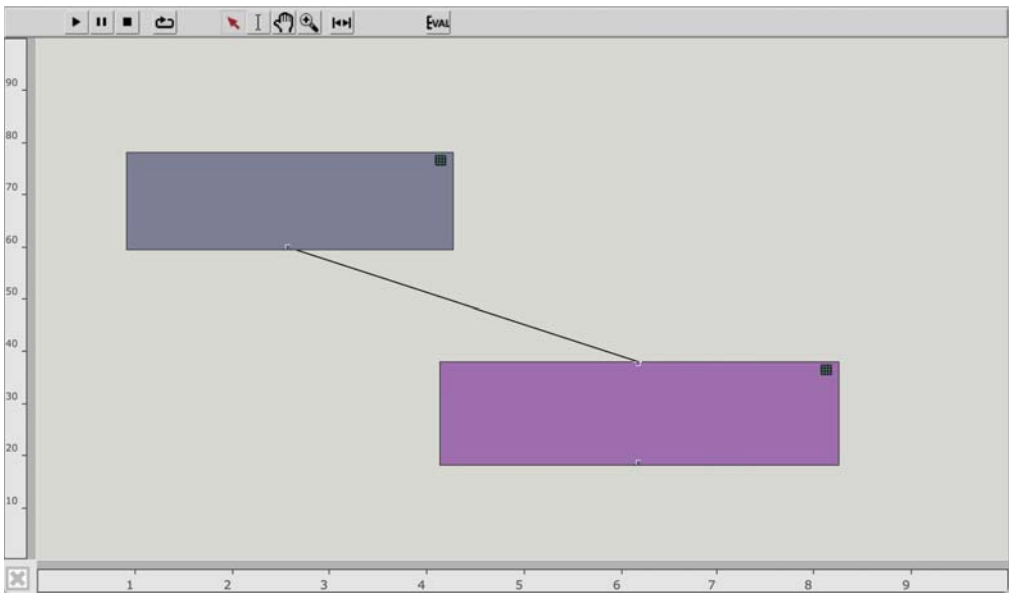


Figura 7: maqueta de OM

A partir de lo expuesto se puede decir que *OpenMusic* es un software muy completo que permite implementar muy variadas técnicas de composición algorítmica, además, si el usuario no encuentra el objeto que se acomode a sus necesidades, es posible crear

objetos propios a partir de un conocimiento básico de LISP. Sin embargo, este software tiene una particularidad que lo hace poco amigable: su carácter críptico. En principio la documentación es muy escasa y muy poco detallada. Lo mismo sucede con la mayoría de las librerías (quizás una excepción sea la librería *OMMatrix* antes mencionada de Pablo Cetta que es expuesta en su libro *Fundamentos de composición musical asistida* (Cetta, 2018)). El compositor que lo utiliza por primera vez se encuentra, casi siempre, ante un panorama un poco desalentador. Muchos objetos arrojan resultados que no son obvios en primera instancia, por ejemplo, si se coloca una clase *Class-Array* y se ejecuta, se obtiene, en la salida que se observa en el *OM Listener*[32], el siguiente mensaje: *OM => #<class-array 40200008CB>*. Por supuesto que esto tiene su razón de ser, pero al no haber una documentación precisa sobre la funcionalidad de cada objeto resulta muy tedioso para el usuario inicial entender qué significa dicho mensaje y cuál es la razón de este objeto.

Otro problema es la interconexión entre *OM* y otros software en tiempo real. Si bien *OM* puede enviar y recibir MIDI y OSC (Open Sound Control) , sus capacidades son limitadas en esta área haciendo que este software no sea apto para su uso en tiempo real. Además, el uso del sistema de maquetas es, si bien poderoso, muy engorroso para crear una cadena de secuencias musicales.

OpusModus

Este software, creado por Janusz Podrazik, al igual que *OpenMusic*, tiene a LISP como lenguaje de bajo nivel, sin embargo, mientras este último utiliza una interfaz gráfica para

32 La ventana Listener es la interfaz principal para Lisp dentro OM. Puede usarse para evaluar expresiones Lisp o leer resultados y mensajes. https://support.ircam.fr/docs/om/om6-manual/co/LispListener.html

interactuar con el usuario (como se muestra en la figura 4), *OpusModus* utiliza un lenguaje propio llamado *OMN*. En el manual del software se lee que “OMN está diseñado como un lenguaje de programación para eventos musicales. No se trata de sonidos se trata de su control y organización en una composición musical. Como un lenguaje de programación lineal en lugar de un pentagrama gráfico, los eventos musicales pueden ser transformados, extendidos y reorganizados por poderosos algoritmos informáticos” (Morgan & Podrazik, 2020). *OpusModus* incorpora algunas herramientas que, si bien existen en *OpenMusic*, son mucho más accesibles y amigables para el usuario. En el lado izquierdo de la figura 8 se observa la interfaz para la programación en lenguaje *OMN,* del lado derecho superior la partitura generada a partir de dicho código y debajo una representación gráfica de las duraciones y las alturas dentro de la partitura. Otra característica importante a destacar es que la documentación de *OpusModus* es excelente. El software posee abundantes archivos en PDF con la descripción de las funciones del programa y con la explicación de cada uno de los algoritmos disponibles.

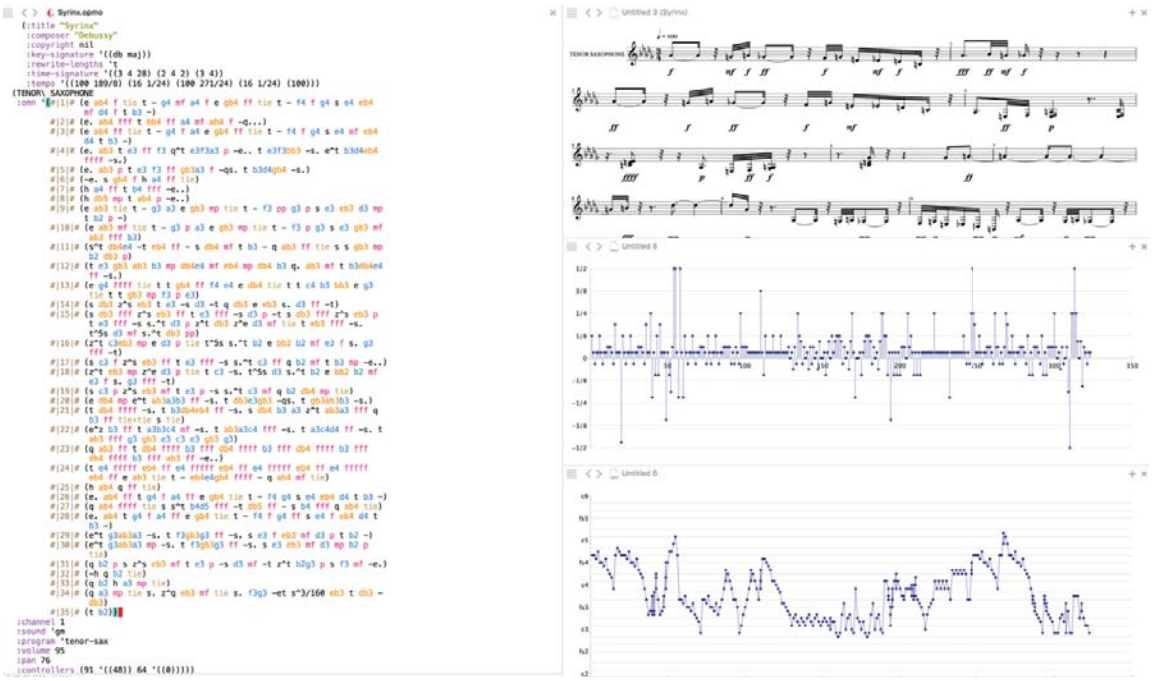


Figura 8: interfaz de OpusModus

Además, cuenta con una web oficial[33] desde donde se pueden descargar ejemplos, videos, tutoriales y participar en un foro especializado donde es posible comunicarse con otros usuarios. En suma, aunque la programación en OMN es más compleja y más intimidante que las que hacen uso de objetos gráficos interconectados, la cantidad y calidad de la documentación hace que cualquier usuario con deseos de aprender este lenguaje tenga todas las herramientas disponibles para tal fin.

Las desventajas que posee este software son las siguientes: al momento de escribir esta tesis, solo está disponible para plataformas Macintosh, esto deja fuera de juego a un gran número de potenciales usuarios[34]. Otro problema es que, si bien se puede utilizar en modo *Live Coding,* no es posible (por lo menos hasta el momento) interactuar con otros programas. Esto permitiría, por ejemplo, sincronizarlo con algún archivo de video o de audio. Por último, al tratarse de una interfaz de código, no es sencillo para el usuario inexperto crear sus propios algoritmos como sí lo es en otros programas que utilizan una interfaz gráfica para la creación de patches.

Nodal

Nodal fue presentando como un proyecto en el Australasian Computer Music Conference 2005[35] por Peter Mcilwain y Jon McCormack. Dicen sus creadores: “El concepto inicial de *Nodal* era crear un método de generación de música a través de una red en la que la salida de un nodo es una nota musical (definida por los parámetros MIDI). El tiempo entre

33 https://opusmodus.com

34 En febrero de 2021 el porcentaje de usuarios de OSX es menor al 10% del total de usuarios de otros sistemas operativos. https://gs.statcounter.com/os-market-share

35 https://computermusic.org.au/conferences/

los eventos de la nota se determina, y se representa, en forma gráfica por la longitud de los arcos entre los nodos.” (McIlwain & McCormack, 2005). Es decir que en el desarrollo de este software el foco fue puesto en crear un sistema que fuera intuitivo y simple de utilizar para el usuario inexperto: “Dado el número de configuraciones y funciones posibles que puede tener una red utilizada para la composición, un software de este tipo puede ser potencialmente demasiado complejo para que un usuario lo configure de manera significativa. Por esta razón, la restricción de diseño más importante fue que el software fuera lo más simple e intuitivo de usar posible.” (McIlwain & McCormack, 2005).

Al observar la interfaz gráfica de usuario que utiliza *Nodal* es posible percatarse inmediatamente de su sencillez, sin embargo, eso no significa que no sea un software poderoso, con muchas posibilidades. Cada nodo (círculos en la pantalla) representa una nota y la distancia entre dichos nodos representa el espacio de tiempo entre notas. La nota tiene tres parámetros: altura, velocity y duración. La altura se expresa en notación anglosajona, es decir C4, D#3, B5, etc., la velocity en valores de 0 a 127 y la duración como porcentaje respecto al inicio de la siguiente nota.

En el ejemplo de la figura 9, cada nota esta configurada con los parámetros por defecto, es decir C3:96:100% (Do central, velocity 96 y una duración del 100%), pero podrían configurarse con una notación relativa a la nota anterior, entonces +1:+10:/2 significaría que cada nodo debe agregar un semitono ascendente, una velocity de 10 y una duración de la mitad del tiempo con respecto a la nota anterior. Otra característica interesante de *Nodal* es que se puede sincronizar con otros programas a través de MTC (Midi Time Code)[36]. Esto significa que puede correr a la par de casi cualquier software dedicado a la

36 https://en.wikipedia.org/wiki/MIDI_timecode

creación multimedia. Es por eso que podría fácilmente ser utilizado para controlar no solo dispositivos musicales sino también programas dedicados al VJ (Video Jockey) y luces, por ejemplo.

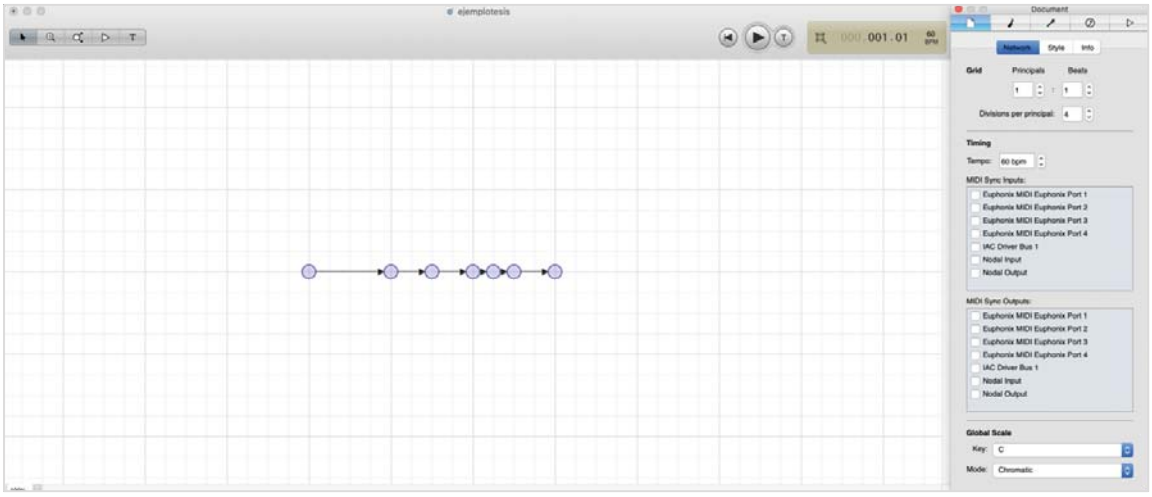


Figura 9: interfaz de usuario de Nodal

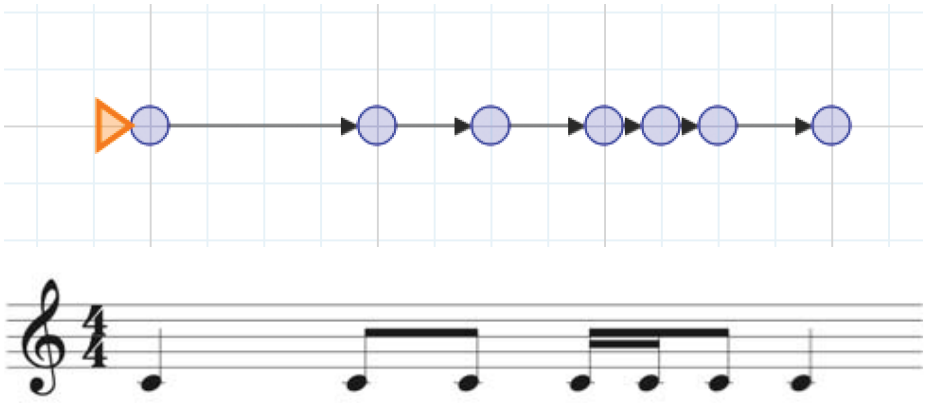


Figura 10: detalle de la forma en que escribimos un ritmo en Nodal

La primera característica de *Nodal* que puede considerarse desfavorable (por lo menos para el trabajo que se propone en esta investigación) es que este tipo de software promueve el uso de loops (o bucles) y en ese sentido es un poco determinante. Esto no

significa que esto no pueda ser evitado, pero la propia interfaz encausa a elegir ese uso. En la figura 11 se observa un típico ejemplo de un patch de *Nodal* donde se aprecia el uso de los loops.

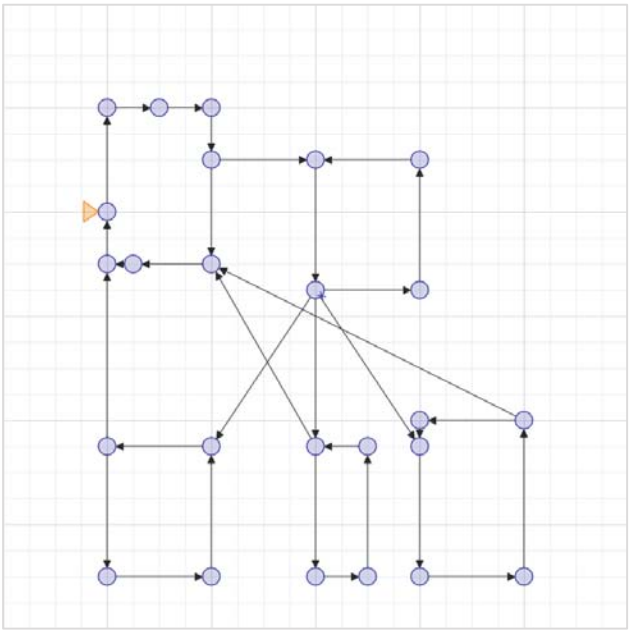


Figura 11: uso de loops en Nodal

Sin embargo, la principal razón por la cual *Nodal* no es apto para esta tesis es que, al ser un software que utiliza una interfaz de alto nivel, no permite crear algoritmos específicos como sí lo hacen *OpenMusic*, *OpusModus*, *SuperCollider*, *Max* y otros; es decir que no se puede, por ejemplo, implementar un sistema de mapas caóticos en este software para controlar un parámetro específico.

SuperCollider

Este software fue presentado por su creador, James McCartney, en el ICMC (International Computer Music Association) de 1996. Acerca de él su creador dice: “La idea de *SuperCollider* es proporcionar un sistema de síntesis de audio en un lenguaje de alto nivel

con un código dinámico, listas, recolección de basura y cierres[37].” (McCartney, 1996). En 2002 fue lanzado como software gratuito (bajo GNU Licencia Pública General). A partir de la versión 3 la arquitectura del mismo se modificó sustancialmente. Hasta la versión 2, *SuperCollider* estaba unificado en una sola aplicación, pero luego se dividió entre la aplicación de lenguaje, llamada el *intérprete*, y un servidor de síntesis llamado *scsynth*. Estos dos se comunican a través de Open Sound Control (OSC). Esto significa que cualquier software que pueda enviar información a través de OSC puede ser utilizado como intérprete. Así se puede comunicar con *Max* o *PureData*[38] u otros entornos creados específicamente para interactuar con SuperCollider como es *Tydal Cycles*[39].

SuperCollider, a diferencia de los programas que se expusieron previamente, no fue creado como una plataforma para la composición asistida por ordenador solamente, sino que fue concebido como un entorno para la síntesis de sonido, el control de dispositivos y el live coding. Esto lo convierte en un software extremadamente poderoso que cuenta con una gran cantidad de usuarios especializados alrededor del mundo. En la figura 12 se aprecia la interfaz de *SuperCollider* donde, a la izquierda, se observa el código de una pieza escrita por Nick Collins en 2007. Se puede ver que la interfaz utiliza un lenguaje de programación propio llamado *sclang* que, si bien no es complejo para aprender, es más intimidante que los lenguajes que utilizan objetos gráficos.

Las razones por las que este software no es el indicado para la creación de la herramienta que propone esta tesis son las siguientes: el hecho de que este utilice un sistema de

37 Refiriéndose a la gestión de la memoria de la computadora

38 https://puredata.info

39 https://tidalcycles.org

código lo hace un poco inaccesible para el usuario inexperto. Otra razón es que no cuenta con una interfaz gráfica desde donde poder ver y escribir partituras, como si la tienen *OpenMusic* y *OpusModus*. Estas características hacen de *SuperCollider* un software poco dúctil para la creación de interfaces que puedan ser usadas, no por el autor del patch, sino por una comunidad de compositores sin experiencia en este tipo de entornos.

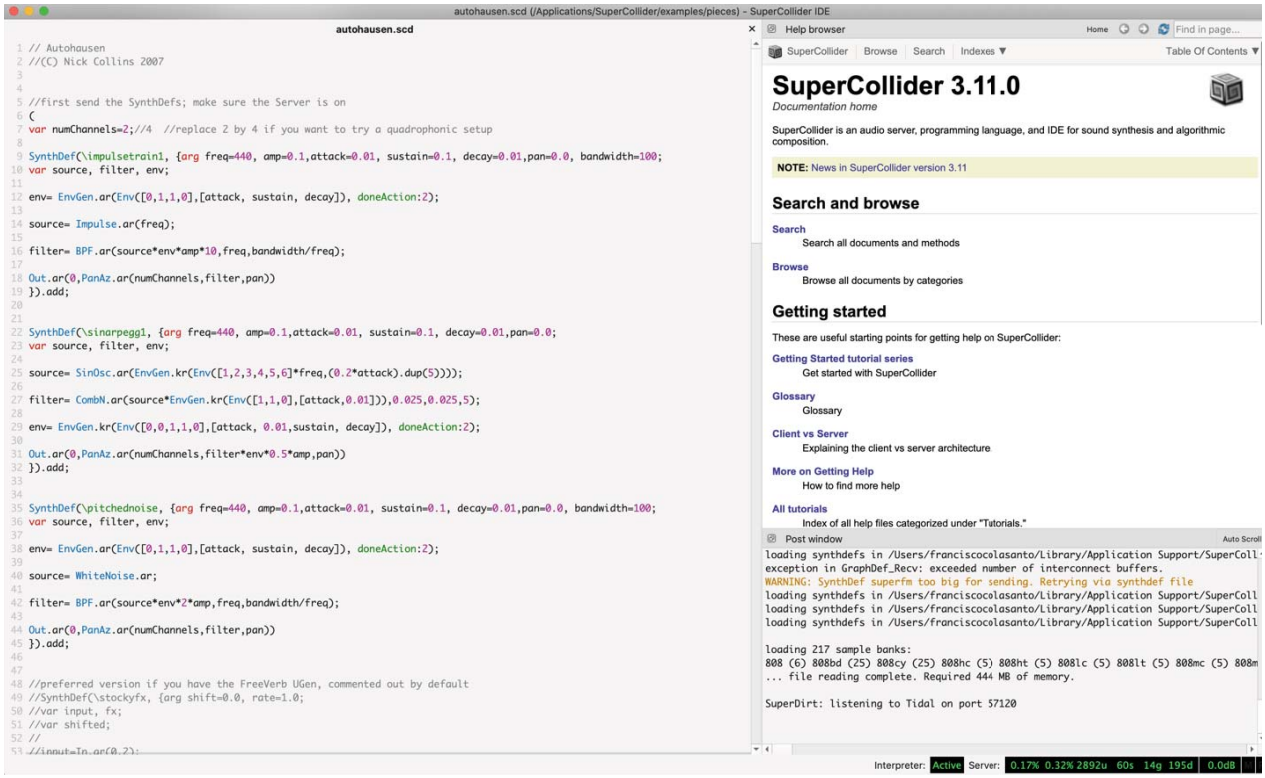


Figura 12: interfaz de *SuperCollider*

Max

En el punto 1.2.3 se habló brevemente de la historia de *Max* y de cómo fue desarrollado por Miller Puckette. Este software, al igual que los otros vistos hasta aquí, con excepción de *Nodal*, es un entorno de programación completo que permite interactuar con datos provenientes de muchas fuentes (MIDI, audio, video, OSC, TCP, puertos seriales, etc.),

además permite crear aplicaciones independientes que pueden ser ejecutadas por usuarios que no posean el software. *Max* ha sido creado para plataformas OSX y Windows[40].

Este software posee tres características que lo hacen más amigable que aquellos basados en código: las funciones están implementadas a través de objetos gráficos, estos objetos se comunican a través de cables que conectan sus salidas (**outlets**) con sus entradas (**inlets**) y posee una serie de interfaces de usuario que permiten controlar y observar datos dentro del software (figura 13).

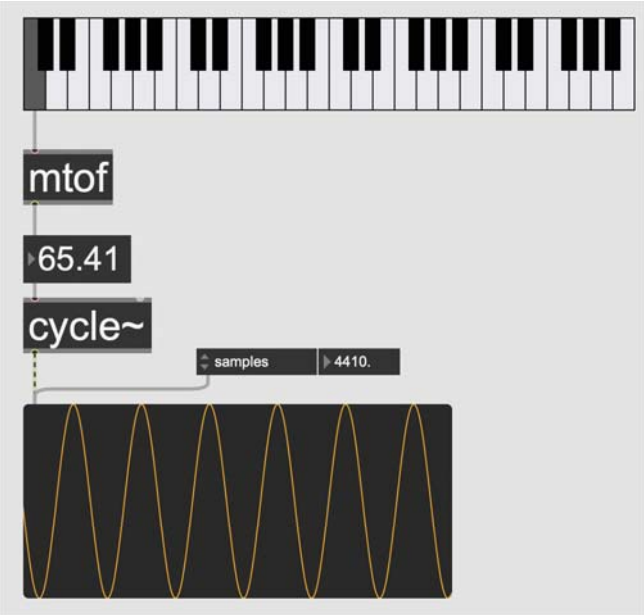


Figura 13: interfaz de *Max*

Otra característica importante de *Max* es que permite trabajar con otros lenguajes de programación como *JavaScript* a través del objeto **js**, *Java* a través de **mxj**, *Gen* a través de gencode dentro de **gen~** (figura 14), *bell* a través de **bach.eval** y otros. Es decir que, a

40 En 1998 se lanzo una versión para plataformas Linux llamado jMax http://jmax.sourceforge.net

diferencia de *SuperCollider* u *OpusModus* donde el usuario solo puede utilizar un lenguaje de programación de código, aquí es posible utilizar objetos gráficos interconectados con objetos que permiten escribir una rutina a través de un script en diferentes lenguajes.

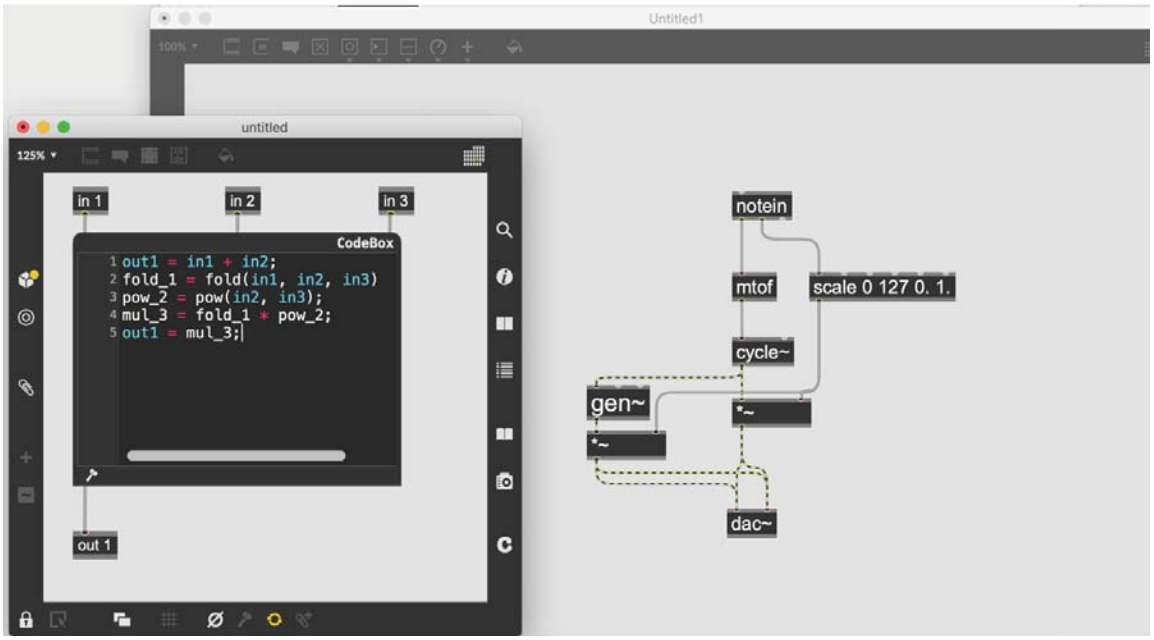


Figura 14: el objeto **gen~** permite utilizar un tipo de script

Max permite que cualquier persona desarrolle sus propios objetos, ya sea en Lenguaje C++ o a partir patches encapsulados. De esta manera existen una gran cantidad de herramientas creadas por desarrolladores alrededor de todo el mundo. Muchas de estas herramientas se encuentran agrupadas dentro de paquetes (librerías) que el software permite descargar desde su propia interfaz (figura 15). Dentro de estas librerías existen tres que son de gran importancia para el trabajo que hemos realizado para esta tesis, estas son *Bach*, *Cage* y *Dada*[41] desarrolladas por Andrea Agostini y Daniele Ghisi (Agostini & Ghisi, 2012) que permiten manipular y mostrar datos musicales en forma de partitura (figura 16).

41 https://www.bachproject.net

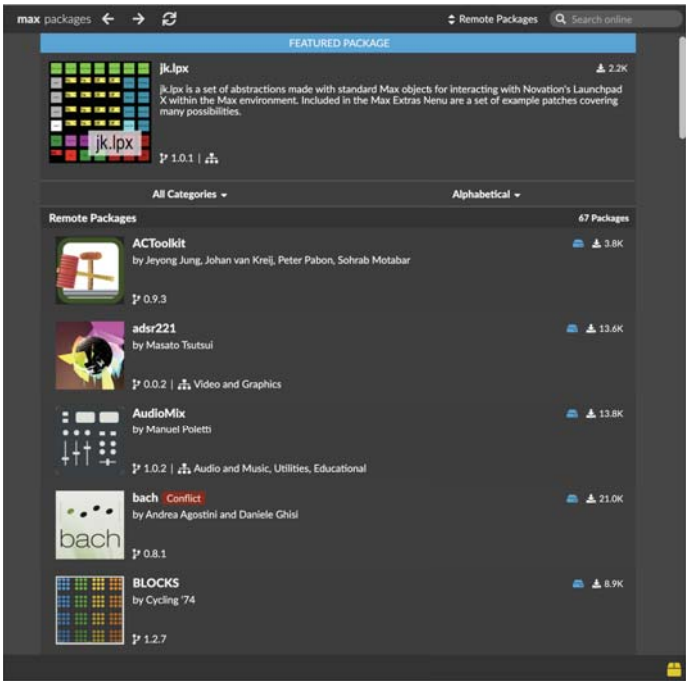


Figura 15: interfaz desde donde se descargan paquetes adicionales

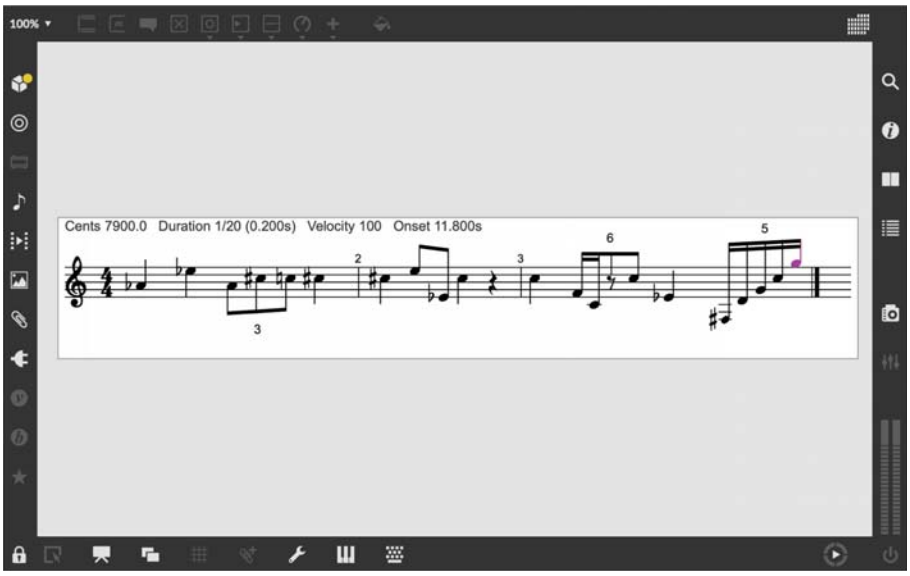


Figura 16: el objeto **bach.score**

Estas tres librerías contienen, además, una serie de objetos inspirados en *OpenMusic* que permiten utilizar muchas de las funciones que dicho software incorpora, pero con una interfaz más amigable y mucho mejor documentada. Hay que destacar que la documentación de *Max* es vasta y también lo es la documentación específica de *Bach*. En cuanto a la interfaz de usuario, *Max* posee muchos objetos que permiten crear una interfaz gráfica donde la programación está oculta, pero con todos los elementos necesarios a la vista que permitan interactuar con el patch creado.

Ableton Live y Max for Live

En 2001 Gerhard Behles y Robert Henke introdujeron en el mercado el programa *Live* (de Ableton). Dicho software es un DAW que incorporó una serie de cambios en la forma de interactuar con el usuario a partir del uso de clips de audio y MIDI que pueden activarse en ciertos momentos específicos de un compás. Estas y otras características lo volvieron un estándar a la hora de tocar en vivo con una computadora (principalmente en el ambiente de la musica electrónica)[42]. Tanto Behles como Henke eran asiduos usuarios de *Max* y muchas de las funcionalidades de *Live* se corresponden con herramientas que ambos habían creado en *Max* para ser utilizadas con su grupo musical Monolake. Por lo tanto la relación entre ambos y los desarrolladores de *Max* siempre fue fluida.

En 2009, Ableton introdujo *Max for Live*[43] en la Expo'74[44] realizada en San Francisco. Esta plataforma permite integrar a *Max* dentro de *Live* (figura 17). A través de esta fusión es

42 https://www.vice.com/en/article/78je3z/ableton-live-history-interview-founders-berhard-behles-robert-henke

43 https://www.ableton.com/en/live/max-for-live/

44 https://cycling74.com/newsletters/max-for-live-presentation-at-expo-74

posible contar con una estación de trabajo que posee todas las ventajas de uno y otro software, es decir, una plataforma de programación dentro de un DAW, así es posible crear patches personalizados que interactúen de manera simple con plugins de instrumentos virtuales, MIDI, audio digital y con pistas donde colocar archivos de audio, MIDI y video. *Max for Live* esta creado de manera tal que permite enviar datos desde *Max* a *Live* y viceversa.

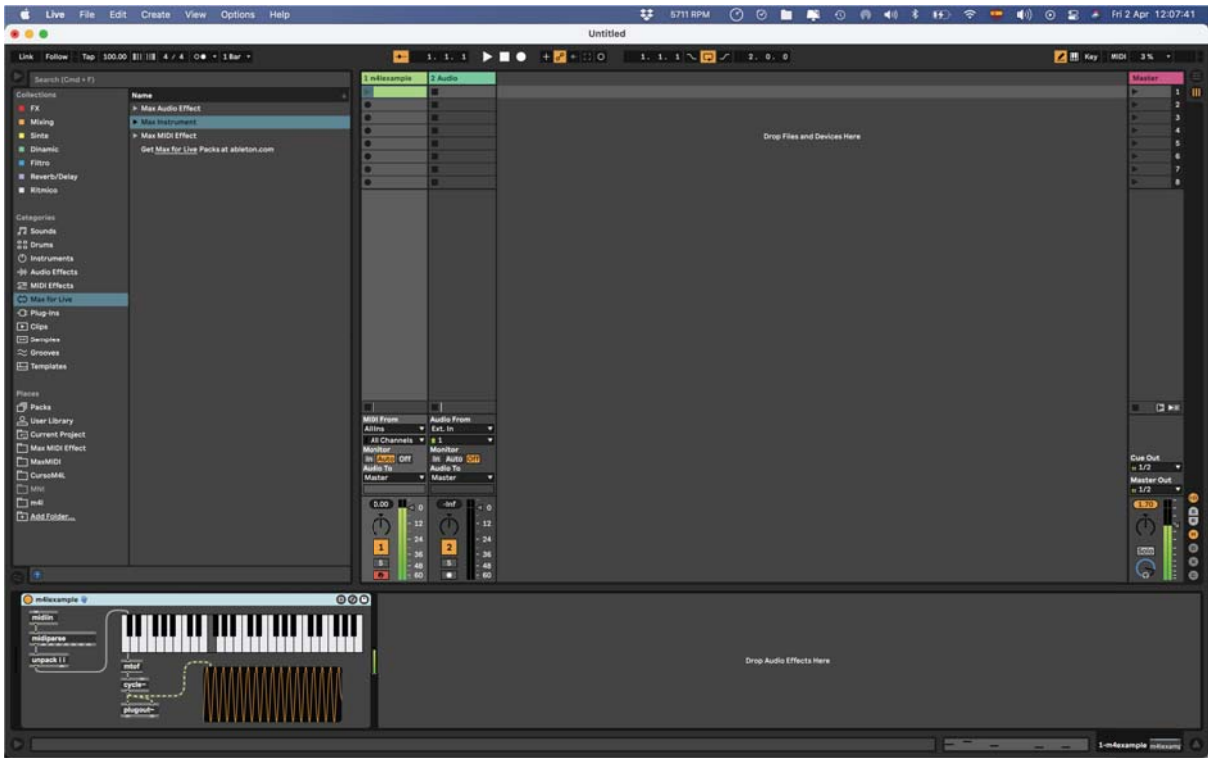


Figura 17: interfaz de *Max for Live*

La comunidad de desarrolladores de dispositivos para *Max for Live* es basta. La más completa (y recomendada por Ableton) es https://maxforlive.com. Allí se encuentra una base de datos con 5034 dispositivos[45] (algunos se descargan de forma gratuita y otros son comerciales) que pueden ser utilizados y, en la mayoría de los casos, pueden ser editados

45 Al momento de escribir esta tesis.

para modificarlos a gusto del usuario o para estudiar la programación a la manera de una ingeniería inversa. Estos dispositivos se encuentran divididos en 16 categorías: *Midi Devices*, *Audio Devices*, *Midi Instruments*, LFO/Modulation, Sequencers, Drum Machine, Sample Glitch, Effects, Jitter/Video, Utility, Experimental/Other, Hardware Control, DJ, Works in Progress, M4L Hack Event, Ableton Push.

1.3 - Justificación

Mi interés por los procesos algorítmicos para la composición se remontan a las primeras obras que he compuesto. De hecho, en el año 2000, como alumno de la Licenciatura en Composición Electroacústica de la Universidad de Quilmes, se estrenó mi obra *El Instante* para orquesta, coro y electrónica, compuesta íntegramente utilizando *OpenMusic.* Desde esos primeros trabajos me he percatado de que los software disponibles para la Composición Asistida por Computadora, al menos los que se mencionaron hasta aquí y que son los más difundidos dentro del ámbito de la composición académica, adolecen de alguna de las características que menciono a continuación: fácil uso, buena documentación, implementación dentro de entornos amigables, clasificación de los datos generados.

El presente trabajo no busca crear una plataforma de programación completa como las que se han expuesto (esto sería demasiado ambicioso) sino crear una herramienta implementada en *Max for Live*, denominada *AMI* (Algorithmic Music Interface), que permite generar alturas, ritmos y dinámicas a partir de distintos algoritmos, almacenarlos dentro del entorno Live, para luego analizar y clasificar dichos datos para que el sistema sugiera relaciones entre ellos.

Las razones por las cuales se hace uso de *Max for Live* como plataforma para la creación de esta herramienta son muchas. La primera, y quizá la menos importante, es que he utilizado *Max* desde 1998 lo que llevó a que haya publicado el primer libro en español sobre programación en dicho entorno llamado *Max/MSP: guía de programación para artistas* (Colasanto, 2010), lo que hace que tenga suficiente experiencia en su uso como para poder aplicar los algoritmos de generación de parámetros que deseo implementar. La segunda es que, al utilizar todas las ventajas de *Max* y *Live* se puede contar con un entorno de programación dentro de una estación de trabajo de audio digital (DAW) lo que hace al sistema muy completo ya que se puede hacer uso de todas las funcionalidades de dicha estación de trabajo como son la edición y reproducción de archivos de audio, video y MIDI. La tercera y más importante es que *Max for Live* puede hacer uso de una serie de librerías externas creadas por diferentes desarrolladores que aumentan las capacidades del mismo. En el caso de *AMI*, la librería *Bach* ha sido fundamental ya que permite contar con interfaces de partituras y permite trabajar de manera muy simple y completa con todo tipo de listas de datos.

1.4 Definición del problema

No existen herramientas intuitivas para la generación y clasificación de parámetros musicales desarrolladas en entornos de uso popular. Dentro de las plataformas que se mencionaron antes (*OpenMusic*, *OpusModus*, *SuperCollider*), es posible implementar casi cualquier algoritmo para tal fin pero presentan, en mayor o menor medida, los siguientes inconvenientes:

- No son intuitivas, como ya se mencionó.

- Requieren del aprendizaje de un lenguaje de programación especifico.
- No están bien documentadas (OpenMusic).
- No poseen herramientas fáciles de utilizar para el análisis y clasificación de los datos generados.
- No son plataformas muy difundidas fuera del ámbito académico o experimental.

Respecto a *Max for Live*, después de examinar la base de datos que se encuentra en https://maxforlive.com se concluye que no hay ningún dispositivo MIDI concebido para la composición algorítmica, sino algunos dispositivos o bien creados para generar datos en tiempo real o bien desarrollados para modificar secuencias previamente creadas, es decir, secuenciadores, arpegiadores o modificadores de los clips escritos. A continuación se detallan algunos de los dispositivos que utilizan procesos algorítmicos y que se encuentran en dicha base de datos:

IOG alt AVLIB fluid rythm 001 1.0 [46]

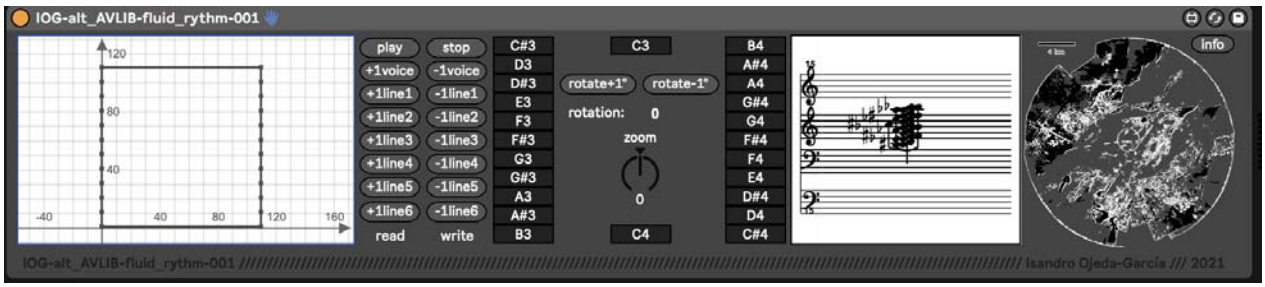


Figura 18: interfaz de IOG[47]

Desarrollado por *Sound Experimentation & New Medias*, permite generar una secuencia de alturas en tiempo real a partir de una interfaz gráfica llamada **dada.bounce**

46 https://www.patreon.com/posts/senm-2021-01-30-46692280

47 Figura utilizada con el permiso Sound Experimentation & New Medias

desarrollada dentro de la librería *Dada*[48].

Esta herramienta no implementa un sistema de generación algorítmica sino mas bien uno geométrico ya que las notas generadas dependerán de la sección del recuadro de la izquierda que sea golpeado por una de las flechas que representa una voz. El tiempo del recorrido de dicha flecha será quien determine el ritmo. No se puede almacenar lo generado a menos que se grabe en tiempo real en un track o clip.

ACDGEN[49]

Figura 19: interfaz de ACDGEN[50]

Desarrollado por Ícaro Ferré, de *Spektro Audio*. Como la mayoría de los dispositivos que permiten generar datos a partir de algoritmos, *ACDGEN* lo hace a modo de arpegiador, donde la probabilidad de notas, duraciones y velocities se estipulan por el usuario aunque los algoritmos utilizados no son expuestos en la herramienta. Sin embargo implementa una función interesante que es la de poder exportar el archivo como clip dentro de un track de Live.

48 https://github.com/bachfamily/dada

49 http://spektroaudio.com/acdgen

50 Con el permiso de Spektro Audio

VLM.Turbulent Sequencer[51]

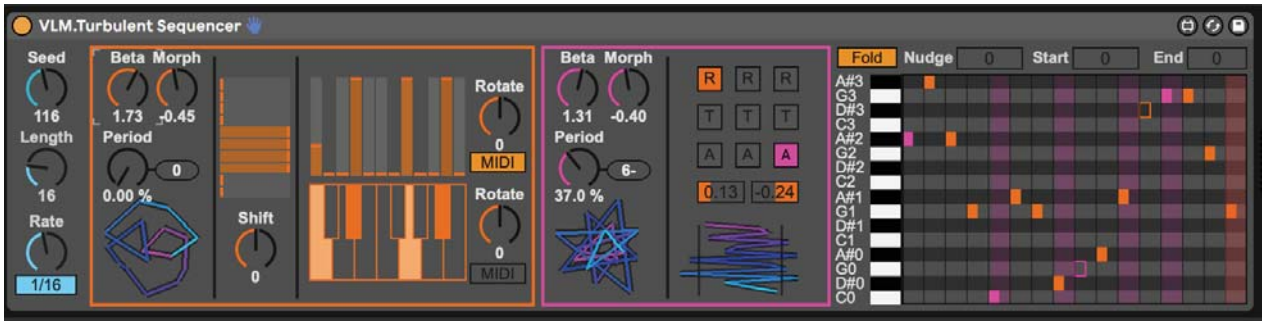


Figura 20: interfaz de VLM.Turbulent Sequencer[52]

Desarrollado por Vulume, este secuenciador genera distribuciones tomadas del análisis de Fourier (FFT) de diferentes tipos de ruido. El autor escribe: “ el parámetro *Beta* establece el color del ruido. En 0 es ruido blanco (una nota no tiene correlación con la nota siguiente). En 2 es ruido marrón (la siguiente nota nunca está lejos, pero sigue siendo aleatoria). En 1 es ruido rosa (una mezcla entre los dos). Más arriba que 2 permitirá que en la restricción de periodicidad domine el ruido. En 10 no hay más aleatoriedad”. Esta herramienta utiliza procesos similares para generar las dinámicas, los silencios y las acentuaciones. La rítmica será la que resulte de dicha interacción.

Los tres ejemplos que aquí se ilustran representan el tipo de desarrollo que se encuentra en la comunidad de *Max for Live* cuando hablamos de procesos algorítmicos. Entonces, si bien *Max for Live* es un software muy difundido dentro del ámbito de la creación sonora y la interpretación en vivo, no existen desarrollos que generen datos simbólicos musicales de manera algorítmica y que permitan una clasificación de dichos datos. En resumen, si bien existe una amplia comunidad de desarrolladores, las herramientas existentes para

51 https://gumroad.com/l/dRhWW

52 Con el permiso de Vulume

Max for Live que implementan ciertas posibilidades de generación de datos a través de algoritmos presentan los siguientes problemas:

- Son, en su mayoría, algún tipo de arpegiador, es decir, no están diseñadas para un uso asincrónico.
- Están muy volcadas a las estéticas de la música popular y especialmente la electrónica, por lo que la generación rítmica es muy simple.
- No utilizan algoritmos complejos como los que podemos encontrar en *OpenMusic*, *OpusModus* o *SuperCollider*.
- No existe una herramienta capaz de analizar y clasificar los datos almacenados en los tracks del software.

1.5 Objetivos

Objetivo general

Crear una herramienta implementada en *Max for Live* llamada *AMI* (Algorithmic Music Interface), capaz de generar perfiles, a partir de diferentes procesos algorítmicos, que puedan ser aplicados a la generación de alturas, ritmos y dinámicas y que pueda clasificar dicho material.

Objetivos particulares

- Diseñar una interfaz de usuario clara y de fácil uso para cualquier persona que tenga conocimientos básicos de tecnología musical.
- Implementar un sistema para que el resultado obtenido de la generación de datos

pueda ser escuchado inmediatamente por el usuario.

- Investigar diferentes algoritmos para la generación de datos para realizar una selección de los más aptos para la herramienta.
- Permitir que *AMI* almacene la informacion generada en forma de clips MIDI.
- Implementar un sistema de clasificación de los datos generados de manera tal que el sistema sugiera posibles relaciones ente las secuencias generadas.
- Investigar diferentes algoritmos para el análisis y clasificación de datos para realizar una selección de los más aptos para la herramienta.
- Programar *AMI* de forma tal que permita, al usuario con conocimientos en programación de *Max for Live*, agregar sus propios algoritmos de generación y análisis.

1.6 Metodología

En primera instancia se creó un base de datos con diferentes algoritmos que permitieran generar los parámetros que se propusieron para este trabajo. Para ello se investigaron las posibilidades que otros software permiten y cómo implementan sus algoritmos. Así se estudiaron las librerías de *OpenMusic* como *Chaos*[53]*, Profile*[54]*, Morphologie*[55] (entre otras), los métodos implementados por *OpusModus* para la generación de secuencias y los objetos de *Max*, propios de esta plataforma y los que otros usuarios han creado para fines similares (como la librería *A-Chaos Lib* de André Sier[56]).

Luego de realizar un recuento de los algoritmos que mejor se ajustan a este trabajo se

53 Mikhail Malt 1998 Paris IRCAM.

54 Jacopo Baboni Schilingi & Mikhail Malt 1998 Paris IRCAM.

55 Jacopo Baboni Schilingi & Frederic Voisin 1998 Paris IRCAM.

56 https://s373.net/code/A-Chaos-Lib/A-Chaos.html

procedió a investigar las ecuaciones necesarias para implementarlos (y que se detallan en el siguiente capítulo). A continuación se realizó un estudio exhaustivo de las librerías ya mencionadas: *Bach*, *Cage* y *Dada* para utilizar, de manera cabal, todas las posibilidades que dichas librerías ofrecen para la creación y manipulación de listas.

Luego se comenzaron a programar diferentes interfaces en *Max for Live*, no con la meta de crear una versión definitiva, sino para entender cuál aproximación podría ser la más completa y amigable para el usuario. A continuación algunas imágenes de estas primeras maquetas.

Figura 21: primera maqueta de interfaz

Todas estas pruebas permitieron llegar a la versión final que se explicará en el punto 1.8 de este capítulo.

Luego se crearon los módulos generadores, programados de manera tal que pudieran entenderse como entidades autónomas para, de esta manera, facilitar la compresión de la programación y para que fuera sencillo, en el futuro, agregar otros generadores a los ya implementados. En la figura 22 se observa un panorama general del sub-patch que contiene a los generadores de alturas, y más abajo un detalle de uno de estos.

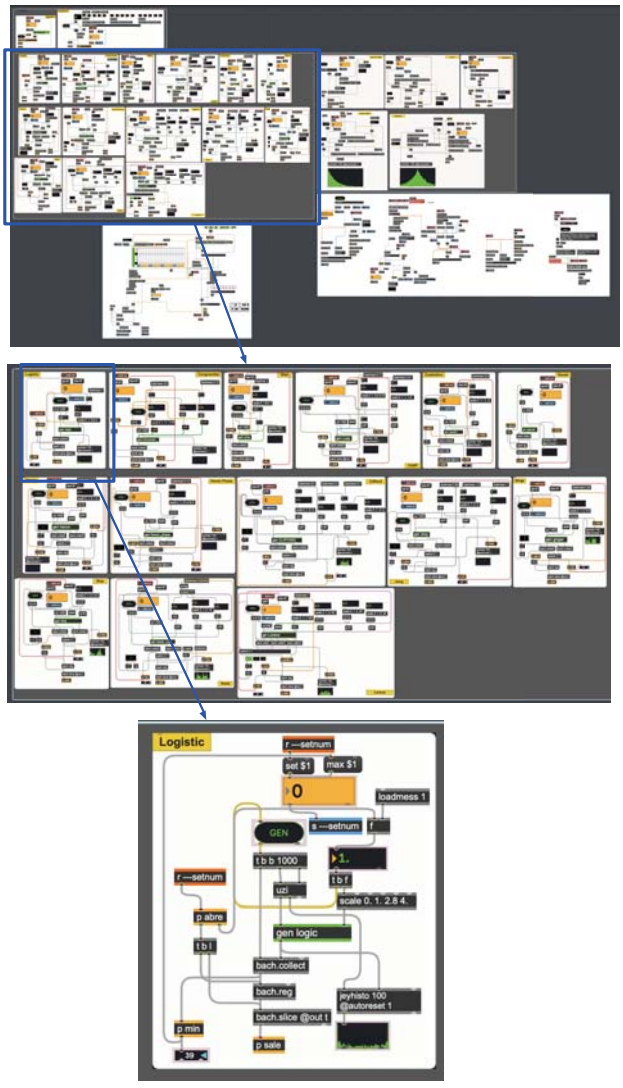


Figura 22: módulos generadores de alturas

A continuación se investigó la manera de escribir, en forma de clip, las secuencias generadas. Para ello se utilizaron los métodos disponibles en el *Live Object Model*[57], que permite la comunicación entre *Max* y *Live*. Esta parte debió programarse dos veces ya que la primera versión fue realizada utilizando *Live 10* y, con la aparición de *Live 11*, la manera de escribir informacion MIDI desde *Max* a *Live* se modificó.

Luego se investigaron las diferentes técnicas de análisis de alturas, ritmos y dinámicas. Como en el caso de los algoritmos de generación de datos, se estudiaron las diferentes ecuaciones y procedimientos necesarios para implementar dichos análisis en *Max*. Aquí también se creó una programación modular encapsulando cada sistema de análisis de manera tal que sea posible aumentar, en el futuro, la cantidad de analizadores disponibles.

Mas tarde se investigaron las diferentes estrategias que podrían utilizarse para clasificar los clips a partir de los datos arrojados por los analizadores. En este sentido se estudiaron los objetos de las librerías para Machine Learning *ml-lib*[58] y *ml.**[59] que implementan algoritmos de clasificación como los de *Adaptive Boosting* , *Gaussian Mixture Model*, *Random Forests* y *K-d tree nearest neighbor.* Finalmente se utilizó la técnica de *nearest neighbor* (vecino cercano) pero haciendo uso el objeto *dada.cartesian* (de la librería *Dada*) ya que éste es completamente compatible con los objetos de la librería *Bach*.

Por último se diseñó la interfaz de usuario y, luego de realizar ajustes a dicha interfaz para

57 https://docs.cycling74.com/max8/vignettes/live_object_model

58 Jamie Bullock and Ali Momeni Copyright © 2020 IRL Labs

59 ©2012-2019 Benjamin D. Smith

mejorar su funcionalidad, se procedió a redactar la presente tesis.

1.7 Generación y clasificación

Como se lee en los objetivos generales, la *generación* de datos y la *clasificación* de los mismos son los ejes centrales se este trabajo. En cuanto a la generación algorítmica ya se mencionan, en la introducción de este capítulo, las diferentes aproximaciones que Laurie Spiegel propone como justificación para el uso de estas técnicas. En cuanto a la clasificación, es importante decir que este trabajo no busca encontrar secuencias idénticas o que musicalmente sean lo más cercanas posibles perceptivamente hablando, o al menos no solo eso, sino que busca clasificar un grupo de secuencias según criterios de análisis que puedan ofrecer pistas al compositor para crear enlaces. Por ejemplo, la entropía se define como una medida del desorden de un sistema (Shannon, 1948), pues bien, si se comparan ciertos grupos de datos y se ordenan de acuerdo a su nivel de entropía (de menor a mayor), las secuencias irían incrementando su nivel de desorden sin que esto signifique que la primera sea idéntica a la segunda. Es decir que esta clasificación puede ser arbitraria pero puede ofrecer herramientas para la cohesión de los datos generados algoritmicamente.

1.8 - *AMI*

Esta herramienta, denominada *AMI*, se encuentra implementada en *Max for Live* dentro de un patch de MIDI, por lo tanto para su uso se debe colocar en un track de ese tipo. Una vez hecho esto, se puede abrir la ventana desde donde se generarán las alturas, ritmos y dinámicas presionando el botón *EDITOR* (figura 23).

Figura 23: interfaz principal de *AMI.*

Al abrir el editor, entonces, se accede a la interfaz que se observa en la siguiente figura:

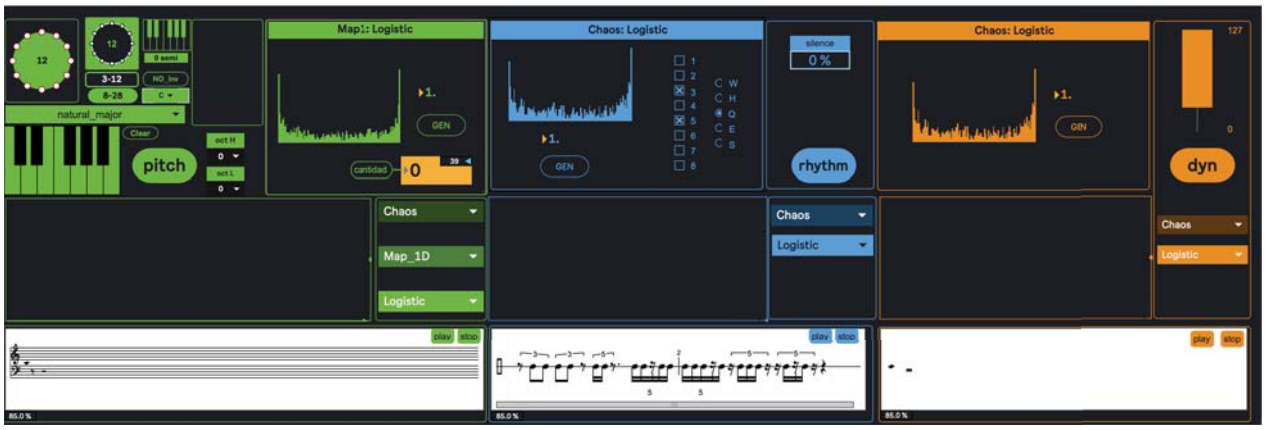


Figura 24: interfaz de usuario del editor

En el sector de la izquierda (verde) se generarán las alturas, en el de centro (azul) los ritmos y en el sector de la derecha (amarillo) las dinámicas. Cada sector será explicado en detalle en el capítulo 3.

Una vez creada la secuencia deseada, esta aparecerá en el pentagrama que se observa en la figura 23. Allí es posible escuchar el resultado haciendo clic en el botón *PLAY* que se encuentra inmediatamente abajo de dicho pentagrama. *AMI* no posee un instrumento virtual propio, por lo que, para escuchar el resultado es necesario agregar a continuación de esta herramienta un instrumento virtual cualquiera como puede ser uno de los varios que se incluyen en la librería de *Live*. Si el resultado obtenido es el deseado, se puede

incrustar el mismo en forma de clip en el track que se escoja en el menú creado para tal efecto.

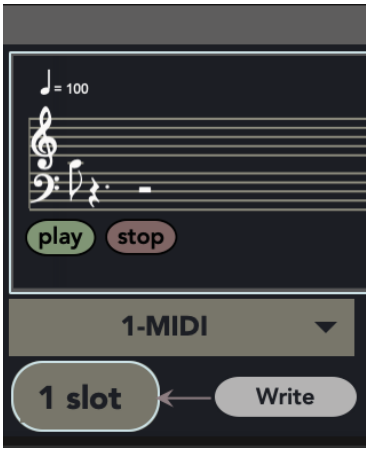


Figura 25: sistema para almacenar clips

En la figura 25 se observa el menú desde donde se escoge el track, el lugar donde se especifica el número de clip donde se almacenará y el botón *Write* que escribirá dicho clip al accionarlo. Se verá esto en detalle en el capítulo 4. Por último, los clips registrados en cualquiera de los tracks MIDI que posea la sesión podrán clasificarse para obtener una jerarquía que indicará cuáles son los más similares de acuerdo a diferentes criterios de búsqueda que el usuario realice (figura 26). Esto se verá en el capítulo 5.

Figura 26: interfaz para la clasificación de los clips generados

Capítulo 2. Generación de parámetros musicales

2.1 - Introducción

Como ya se especificó, la presente investigación se centra en la factibilidad de crear una herramienta, denominada *Algorithmic Music Interface* (*AMI*), para la generación, mapeo, análisis y clasificación de parámetros musicales, específicamente, altura, ritmo (ataques y duraciones) y dinámicas. Este capítulo expone los procedimientos y fórmulas que se utilizaron para la generación de valores discretos, que luego serán mapeados, analizados y clasificados.

Las técnicas algorítmicas que se pueden utilizar para tal fin son numerosas (Fernández & Vico, 2013; Langston, 1988; Miranda, 2001; Nierhaus, 2009; Siphocly, 2021). Las que se han escogido para este trabajo se encuentran entre aquellas que no necesitan del análisis de un corpus previo para ser implementadas. Se dividieron a estas en cinco categorías generales: *gramaticales* (de contexto libre, de contexto sensitivo y estocástico), *caóticas* (mapas de una, dos y tres dimensiones), *aleatorias* (random walk, lineal, triangular, exponencial, bilateral, gauss y beta), *genéticas o evolutivas* (técnicas de selección, técnicas de cruzamiento y técnicas de mutación) y *autómatas celulares* (dimensión 1, game of life). Entre las técnicas que requieren de un análisis previo, y que por lo tanto quedaron fuera de este trabajo, se pueden citar a aquellas que Nierhaus engloba dentro de la esfera de inteligencia artificial (Nierhaus, 2009): *redes neuronales artificiales*, m*achine learning* y c*adenas de Markov*. Tampoco se incluyó una técnica denominada c*onstraints* que, aunque no implica utilizar un sistema de análisis previo, su implementación a nivel de la interfaz de usuario es muy compleja, sin embargo será incorporada en un futuro cercano.

2.2 - Gramaticales

La piedra fundacional de esta técnica denominada *método gramatical*, se encuentra en los trabajos que Noam Chomsky (1928) realizó en la década de 1950 sobre lingüística. En este sentido es especialmente importante su libro *Syntactic structures*[60] (Chomsky, 1957). El método gramatical busca descomponer una frase compleja en sus partes constitutivas, es decir las palabras, y crear así una teoría de la estructura del lenguaje sin referencia a un idioma específico. De esta manera el análisis gramatical permite construir una estructura jerárquica dentro del lenguaje. De aquí se desprende que, como la mayoría de las formas musicales también poseen dichas jerarquías, muchos compositores hayan buscado relacionar el campo de la lingüística y el de la música (Gillick, Tang, & Keller, 2010; Holtzman, 1981; McCormack, 1996; C. Roads & Wieneke, 1979). Las palabras y oraciones que se utilizan al hablar y escribir se generan a partir de reglas de escritura que se aplican repetidamente, por lo tanto, dichas reglas pueden ser aplicadas a la generación de secuencias de símbolos musicales. Uno de los escritos más influyentes sobre este uso es el libro *A generative theory of tonal music* (Lerdahl & Jackendoff, 1983), allí los autores dicen: “La teoría de la lingüística generativa es un intento por caracterizar qué es lo que los humanos saben cuando saben cómo hablar un determinado lenguaje, habilitándolo a entender y crear un gran número indefinido de sentencias, la mayoría de las cuales no había escuchado previamente”. Si se aplica entonces un sistema de reglas recursivas para definir un lenguaje capaz de crear expresiones coherentes dentro de un contexto dado, podemos hablar de una gramática generativa (Nierhaus, 2009). La propuesta de Chomsky es generar nuevas secuencias de palabras a partir de un sistema denominado *reglas de*

60 En el comienzo de la introducción escrita para la segunda edición, David W. Lightfoot escribe: “Estructuras sintácticas de Noam Chomsky fue la bola de nieve que comenzó la avalancha de la “revolución cognitiva” moderna”.

reescritura, donde los símbolos que se escriben a la izquierda dentro de una expresión dada son reemplazados por los de la derecha. El ejemplo que propone Chomsky es el siguiente:

S → SN + SV

SN → A + Sus

SV → Verb + SN

A → el, la, los, las

Sus → hombre, pelota, etc.

Verb → golpea, toma, etc.

Donde *S* = Sentencia, *SN* = Sintagma Nominal, *SV* = Sintagma Verbal, *A* = Artículo, *Sus* = Sustantivo, *Verb* = Verbo. Entonces, al analizar la sentencia “El hombre golpea la pelota”, es posible diseccionarla como se muestra en la figura 27.

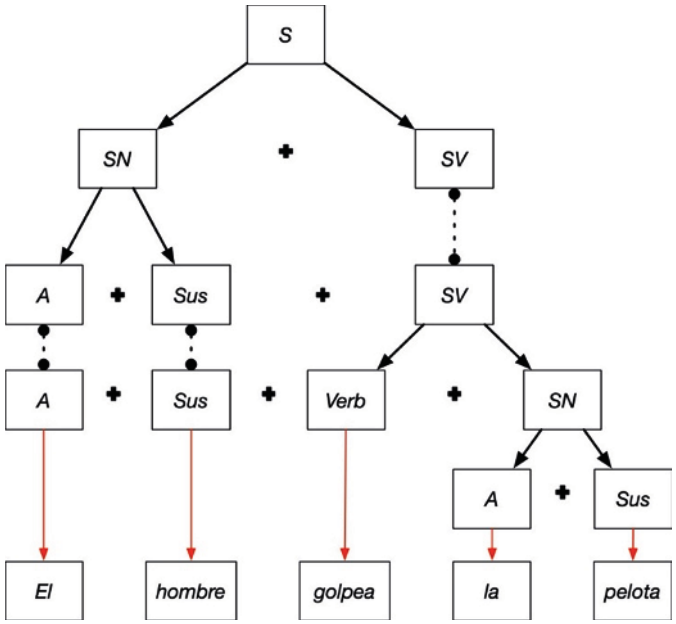


Figura 27: análisis de la sentencia “El hombre golpea la pelota”

Dentro de este sistema deben distinguirse los símbolos que pueden ser re-escritos (símbolos no-terminales), de los que no pueden serlo (símbolos terminales). Es común representar a los símbolos no-terminales con letras en mayúscula *A*, *B*, *C*, etc. y los terminales en minúsculas *a*, *b*, *c*, etc., mientras que el símbolo de flecha (→) significa "se define como" y el símbolo | significa "o" (Miranda, 2001). El proceso comienza con un símbolo no-terminal que se reemplaza por otro asociado a este a través de la flecha (del lado derecho) y que puede ser un símbolo terminal o no. Si no lo es, el proceso continúa, en cambio si es un terminal se detiene. Un ejemplo de esto:

S → A | B

A → aA | a

B → bB | b

Es decir que *S* se define como *A* o *B*, *A* se define como *aA* o *a*, y *B* se define como *bB* o *b*. En la figura 28 se observa[61] un camino posible que puede recorrerse a partir de estas reglas.

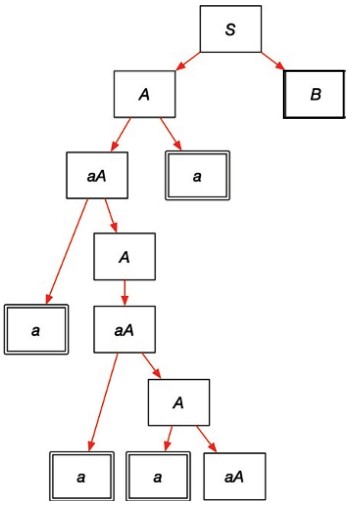


Figura 28: recorrido posible a partir de las reglas de la figura 25

61 Los recuadros dobles se utilizan aquí como indicación de que hemos llegado a un símbolo terminal.

Es decir, S → A → aA → aaA → aaaA…

Este método busca generar una serie datos a partir de un grupo de reglas gramaticales definidas como α → σ, donde σ puede derivarse de α. La figura 29 presenta un ejemplo más complejo[62]. Dadas las siguientes reglas:

S → aSGH | aGH

HG → GH

aG → ab

bG → bb

bH → bc

cH → cc

se puede construir el siguiente camino:

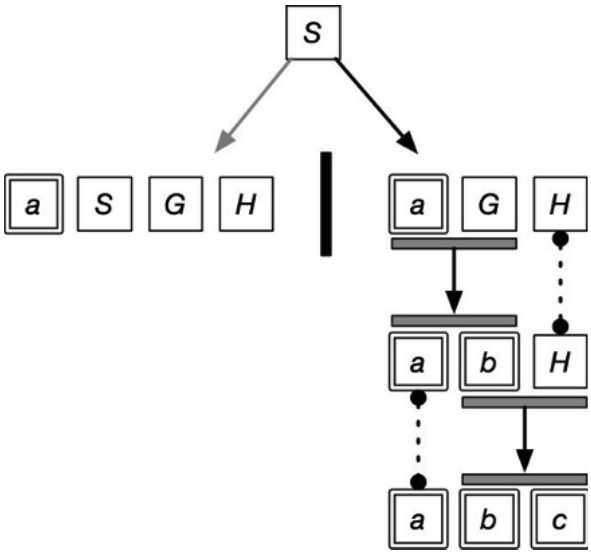


Figura 29: recorrido posible

[62] Recordar que las mayúsculas son no-terminales y las minúsculas son terminales.

Es decir, S → aGH → abH (por regla 3) → abc (por regla 5) y a partir de allí ya no se puede continuar porque todos los símbolos son terminales.

Si se toma otro camino se podría obtener entonces un recorrido como el de la figura 30:

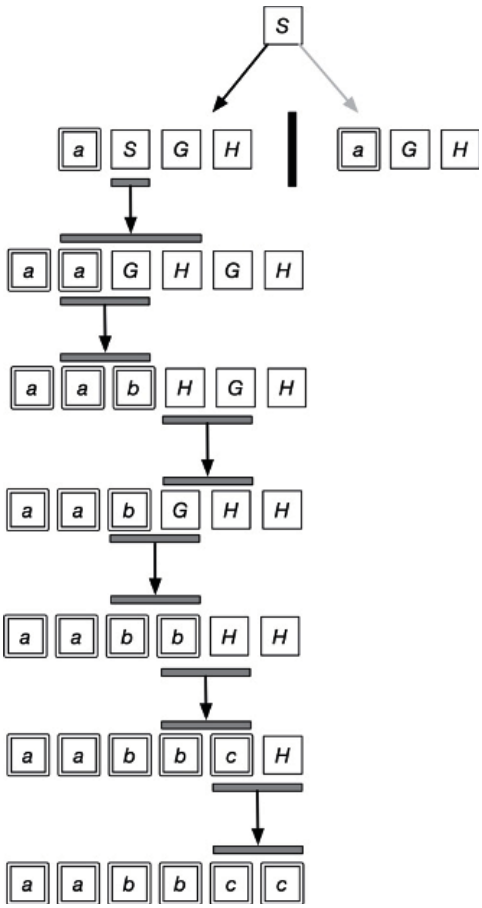


Figura 30: recorrido posible

Las reglas de producción gramaticales se pueden formalizar de la siguiente manera: *N*, *T*, *P*, *S* donde:

1) *N* es un conjunto de nodos no terminales.

2) *T* es un conjunto de nodos terminales.

3) *P* es un conjunto de reglas de producción en forma de α → σ.

4) *S* es designado como el símbolo inicial de la gramática.

De esta manera se puede entonces definir a *G1* (*N, T, P, S*) como ([*A*], [*a, b, c*], *P, A*), donde [*A*] es el nodo no terminal, [*a, b, c*] son los nodos terminales, las reglas de *P* son *A* → *aA*, *A* → *abc* y *S* = *A*. De esta manera se puede construir:

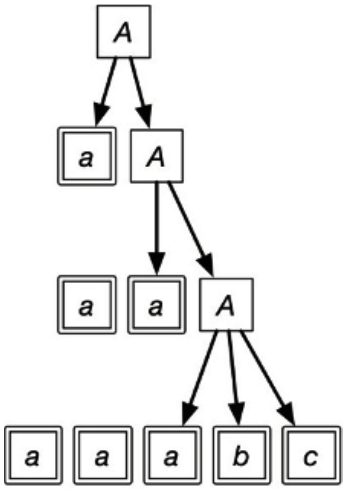


Figura 31: recorrido a partir del último ejemplo.

A este tipo de gramáticas, en las que todas las reglas de producción contienen un solo nodo no-terminal del lado izquierdo, se las conoce como *context-free grammars* (Miranda, 2001).

El método desarrollado por Chomsky es mucho más complejo que la simple introducción expuesta aquí y su explicación cabal excede la meta propuesta para esta tesis. Sin embargo, si bien hay ejemplos de obras que utilizan algoritmos que implementan estas técnicas para la composición[63], es muy complejo crear “a mano” reglas gramaticales que

63 Ver la tabla 1 en (Fernández & Vico, 2013)

generen ideas musicales[64]; una forma de solucionar esto es analizar una serie de obras ya compuestas para obtener de allí las reglas que serán utilizadas. Como ejemplo de un sistema de análisis-generación que involucra métodos gramaticales (entre otras técnicas más complejas) se puede citar el sistema *EMI Experiments In Music Intelligence* creado por David Cope (Cope, 1987). Allí, el mencionado compositor se puso como meta crear un antagonista de sus propios sesgos musicales y desarrollar un mecanismo para estudiar el potencial que tiene el definir a la música en términos lingüísticos.

Como ya se mencionó, la herramienta desarrollada como parte de este trabajo de tesis doctoral pretende generar perfiles de melodía, duraciones y dinámicas que no requieran del análisis previo de un corpus musical. Es por esta razón que no se ha buscado implementar este método. Sin embargo, existe una técnica gramatical llamada *L-System* que puede aplicarse sin utilizar dicho análisis.

2.2.1 - L-System

Una alternativa a las reglas gramaticales de Chomsky aplicadas a la composición algorítmica es el uso de los *Sistemas de Lindenmayer* o *L-System*. Przemyslaw Prusinkiewicz (Prusinkiewicz, 1986) explica que, si bien los *L-System* generan una sucesión de símbolos que sustituyen repetidamente al predecesor por un sucesor dado, difiere de la gramática de Chomsky en que, mientras que en esta las reglas gramaticales son aplicadas de forma secuencial, las reglas de *L-System* son aplicadas a todos los

64 Aunque puede citarse el trabajo de Bel y Vecchione (Bel & Vecchione, 1993) donde hacen uso de un software de su autoría llamado Bol Processor el cual, según sus palabras, “se utiliza para la composición musical e improvisación con MIDI en tiempo real, archivos MIDI, Csound y opciones de salida de texto. Produce música con un conjunto de reglas (una gramática compositiva) o partituras de texto que se pueden escribir o capturar desde un instrumento MIDI."

símbolos de una cadena dada al mismo tiempo. El biólogo teórico húngaro Aristid Lindenmayer (1925 - 1989), utilizó este método como una manera de descripción formal del crecimiento de organismos vivos (Lindenmayer, 1968). Los símbolos comúnmente usados para definir un *L-System* son:

υ = alfabeto que consta de un conjunto finito de símbolos tales como [*a, b, c, d*].

υ^* = conjunto de todas las cadenas posibles de υ (incluyendo el conjunto vacío $\varnothing$). Por ejemplo: *ababa*, *caaaadb*, etc.

$\upsilon+$ = conjunto υ^* excluyendo el conjunto $\varnothing$.

ω = axioma o iniciador ($\omega \in \upsilon+$).

P = conjunto finito de reglas. *P* se representa de la siguiente forma: $\alpha \rightarrow \sigma$, con el predecesor $\alpha \in \upsilon$ y el sucesor $\sigma \in \upsilon^*$

Según su implementación podemos distinguir diferentes clases de *L-System:* de contexto libre, de contexto sensitivo y estocástico (Manousakis, 2006).

De contexto libre

La clase más simple de *L-System* es aquella denominada *de contexto libre* o *D0L-System* (Prusinkiewicz & Lindenmayer, 2012). Dada la siguiente definición:

υ = [*a, b*].

ω = *a*

$P = \quad a \rightarrow ab$

$\qquad\quad b \rightarrow a$

se puede construir:

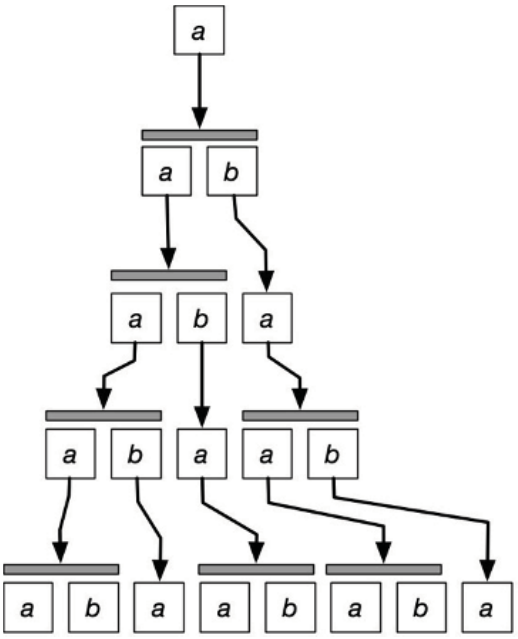


Figura 32: ejemplo de D0L-System

Este tipo de *L-System* es denominado de contexto libre y determinístico porque las substituciones no toman en cuenta el entorno donde se producen (es decir los símbolos que preceden o anteceden) y siempre se obtiene, para el mismo dato de entrada, el mismo resultado de salida.

De contexto sensitivo: 1L-System

Aquí se toma en consideración no solo el símbolo en cuestión sino también su sucesor o antecesor. A este sistema se los denomina *1L-System*. Es decir que las reglas serán:

contexto izquierdo < predecesor > Ø → sucesor ò

Ø < predecesor > contexto derecho → sucesor

Un ejemplo sería:

υ = [*a, b*].

ω = *ab*

P = *ba* → *aba*

ab → *bab*

Así se obtiene:

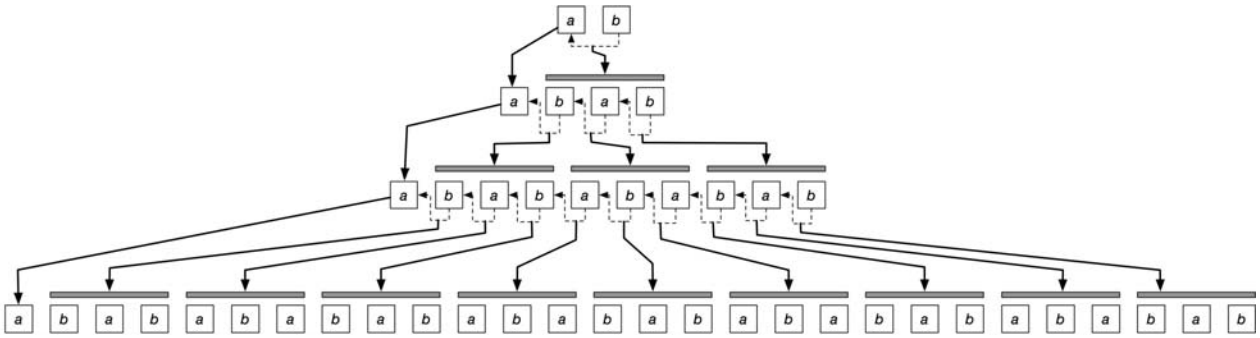


Figura 33: ejemplo de *1L-System*

Es habitual considerar al primer elemento (que no tiene un vecino a la izquierda) como si se definiera a si mismo, por lo tanto, genera el mismo símbolo. Sin embargo, en un caso como el de la figura 33, este tipo de gráfica se vuelve poco inteligible, por lo que es más claro decir que el resultado será:

n_0 = *ab*

n_1 = *abab*

n_2 = *abababababab*

n_3 = *abababababababababababababababab*

De contexto sensitivo: 2L-System

Por último, es posible utilizar un sistema que tome en consideración los símbolos que se encuentran tanto a la izquierda (antecesor) como a la derecha (sucesor) del símbolo escogido. Este sistema se denomina *2L-System*:

υ = [*a, b, c*].

ω = *aaababbbabcb*

P = *aaa* → *abc*

bcb → *bab*

bab → *aaa*

Aquí hay que considerar el símbolo que completará las cadenas incompletas. A esto se lo denomina generalmente como *padding* o simplemente *pad*. Aplicado a la cadena inicial de este ejemplo:

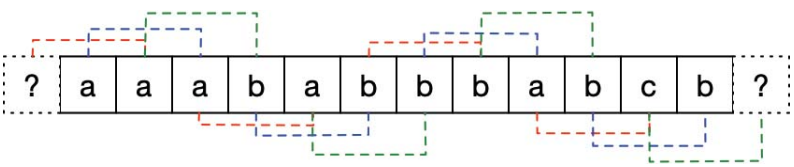


Figura 34: cadena inicial del presente ejemplo

Entonces ¿qué valores tienen los casilleros marcados con el signo de interrogación?. Una técnica es agregar una copia del valor contiguo (en el caso del primer casillero será una *a* y en el del último será una *b*). Otra técnica es completar el primer casillero con el valor del último y el último con el del primero, lo que resultaría *baaababbbabcba*.

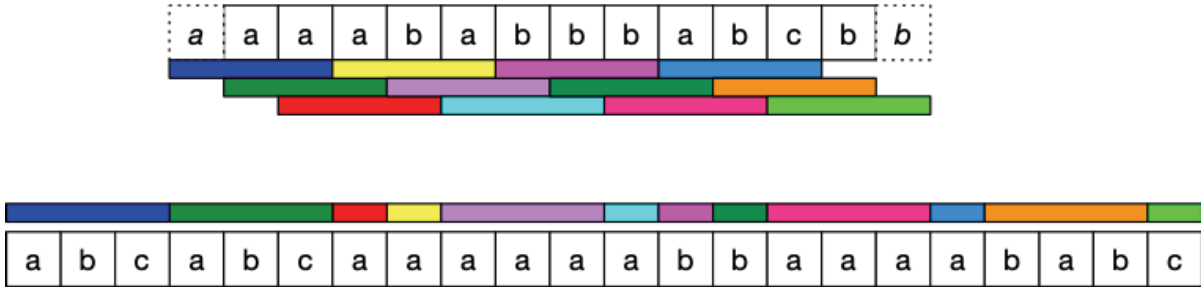


Figura 35: padding

L-System estocástico

Los sistemas vistos hasta aquí fueron todos del tipo determinístico, es decir, que para el mismo conjunto de reglas se obtiene siempre el mismo resultado. Esto no debe ser necesariamente así. Es posible crear un *L-System* no-determinístico si el conjunto finito de reglas utiliza un porcentaje de probabilidad de transición $P = \alpha \xrightarrow{0.5} \sigma$. En este caso, entonces, la probabilidad de que α sea reemplazado por σ es de un 50%. Es importante decir que aquí la suma de las probabilidades de un mismo α deben siempre ser igual a 1. Si se repite el ejemplo del *D0L-System* pero utilizando un sistema estocástico definido como:

$v = a, b$
$w = a$

$$p = a \xrightarrow{0.5} ab$$
$$a \xrightarrow{0.5} ba$$
$$b \xrightarrow{0.7} a$$
$$b \xrightarrow{0.3} bb$$

Se obtendrá:

n_0 = a, n_1 = ab, n_2 = baa, n_3 = aabba, n_4 = abbaabbab ó

n_0 = a, n_1 = ba, n_2 = aba, n_3 = ababa, n_4 = abbbbaaab

Representaciones gráficas de L-System

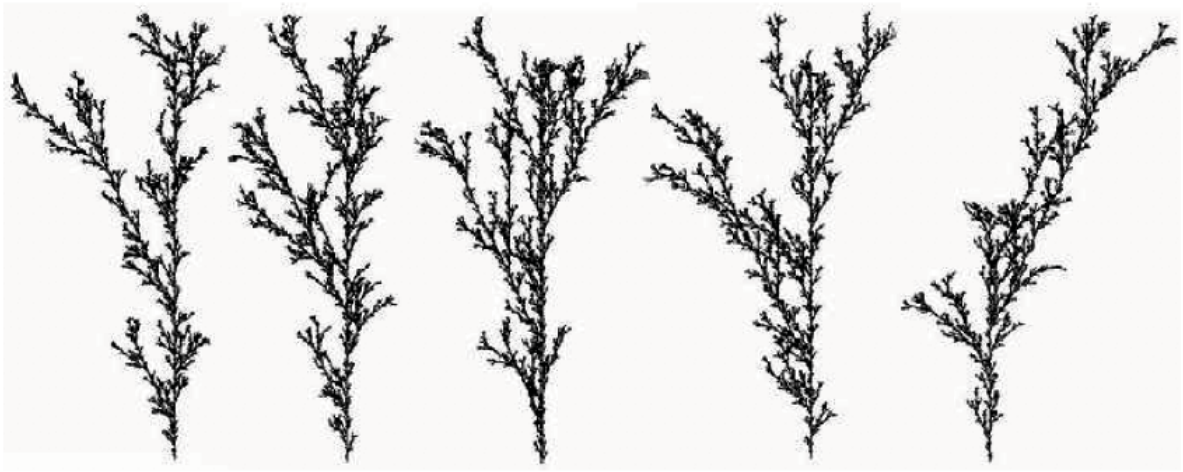


Figura 36: dibujos semejantes a plantas generados con L-System

Como ya se mencionó, este método fue desarrollado como una manera de descripción formal de crecimiento de organismos vivos (específicamente vegetales). Por lo tanto, al graficar una secuencia de símbolos generados por algunos tipos de *L-System* se obtienen imágenes que se asemejan mucho a las de una planta (figura 36), mientras que otras variantes generan formas fractales (figura 37).

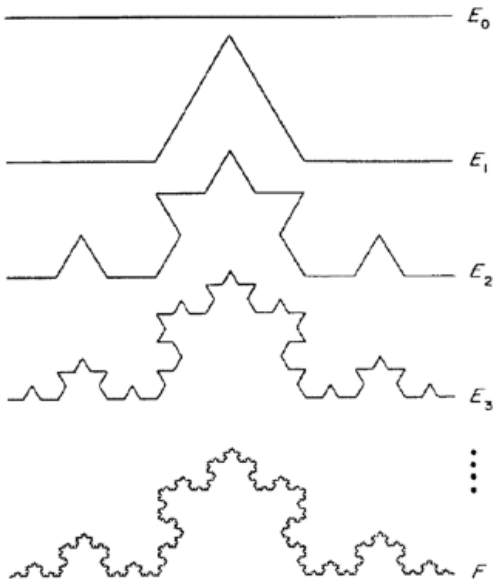


Figura 37: fractales generados con L-System

Para crear estos gráficos, la técnica más utilizada es la denominada *gráfico de tortuga*[65] que consiste, en su forma mas básica, en interpretar ciertos símbolos como directrices para trazar “el camino de una tortuga”. Por ejemplo:

F = avanzar un paso de un largo dado.

+ = rotar a la izquierda en un ángulo θ.

- = rotar a la derecha en un ángulo θ.

65 Originalmente creados para el lenguaje de programación LOGO

θ = ángulo de rotación.

I = profundidad de la iteración.

Entonces con estas reglas:

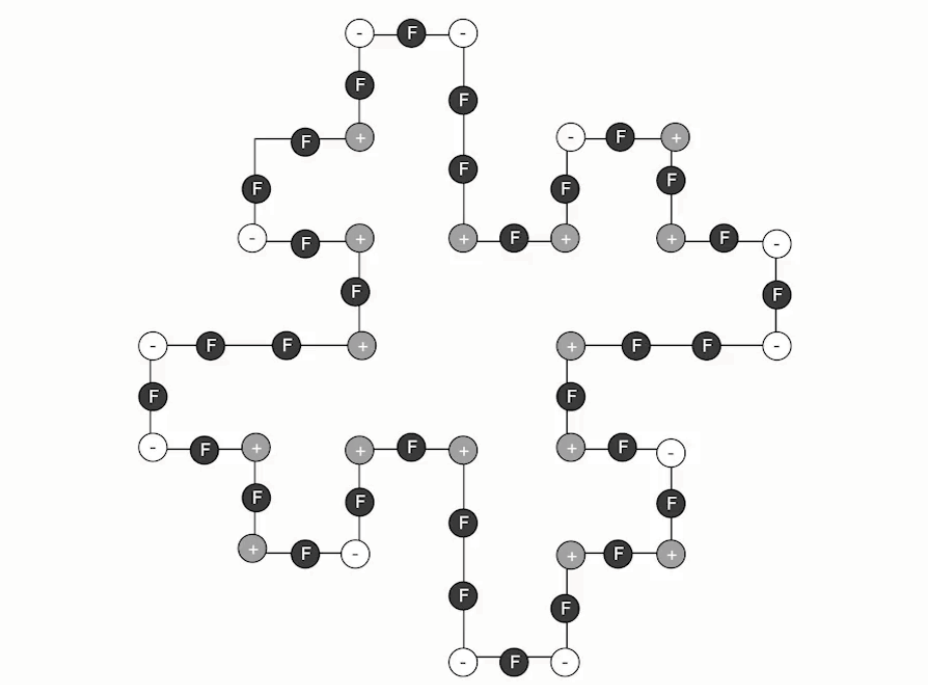
ω = F+F+F+F

P = F → F+F-F-FF+F+F-F

θ = 90º

I = 1

Se obtendría: F+F-F-FF+F+F-F + F+F-F-FF+F+F-F + F+F-F-FF+F+F-F + F+F-F-FF+F+F-F (video 1).



Video 1: gráfico creado a partir de las reglas propuestas arriba

Si se realizan cuatro iteraciones con las mismas reglas se obtendría una figura como la que se muestra en la figura 38.

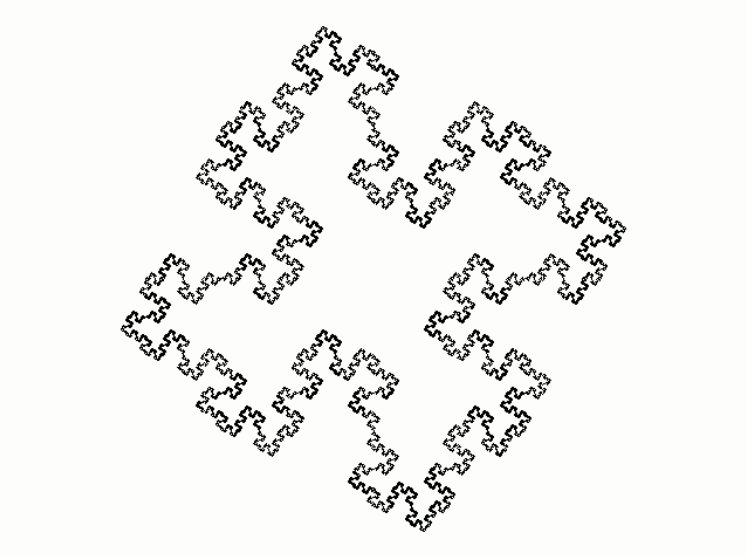


Figura 38: resultado de iterar las mismas reglas cuatro veces

Pueden formarse gran cantidad de figuras interesantes que a partir de un grupo de reglas específicas[66].

2.3 - Caóticos

Existen dos clases generales de procesos aleatorios, los que utilizan el resultado previo como entrada para obtener el próximo valor y los que son independientes de sus resultados previos (Dodge & Jerse, 1985). Entre los primeros se encuentran los sistemas caóticos.

2.3.1- Caos

David P. Feldman define: “Caos es un fenómeno que se encuentra en las ciencias y en las

66 Ver por ejemplo http://paulbourke.net/fractals/lsys/

matemáticas en el cual un sistema determinista (basado en reglas) se comporta de manera impredecible" (Feldman, 2012). Por sistema determinista se refiere a una función matemática que está determinada por sus datos de entrada, es decir, que a igual dato de entrada, se obtiene siempre el mismo resultado. A una función que no cumple con esta característica se la llama estocástica. Un tipo especial de función es aquella que utiliza, para calcular su salida, una iteración, es decir una función que se aplica de manera repetida utilizando su salida como entrada del próximo cálculo. Por ejemplo:

$$x_{n+1} = {x_n}^2$$

Ecuación 1

La letra *n* representa una posición o índice dentro de una lista de valores, así, por ejemplo, si tomamos los primeros ocho valores de una serie de Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21, cada *n* corresponde a uno de los valores de la lista:

$$x_0 \to 1,\ x_1 \to 1,\ x_2 \to 2, x_3 \to 3, x_4 \to 5,\ x_5 \to 8,\ x_6 \to 13,\ x_7 \to 21$$

La función debe tener siempre una condición inicial, y esa condición afectará de forma significativa el resultado obtenido. Este resultado será una secuencia de valores a la que se denomina *órbita* o *itinerario*. Por ejemplo, si a la ecuación 1 se le asigna un valor inicial de 1 entonces:

$$x_{n+1} = {x_n}^2$$

$$x_0 = 1$$

$$f_{x+1} = x_0 \to 1,\ x_1 \to 1, x_2 \to 1,\ x_3 \to 1,\ x_4 \to 1,\ x_5 \to 1,\ x_6 \to 1,\ x_7 \to 1$$

Ecuación 2

Ya que:

$$\left(\left(\left(\left(\left(\left(1^2\right)^2\right)^2\right)^2\right)^2\right)^2\right)^2 = 1$$

Ecuación 3

Lo mismo sucedería si $x_0 = 0$. En estos casos, al valor 1 y al 0 se los denomina puntos fijos de la función, en cambio si $x_0 = 2$ se obtiene entonces:

$$x_{(n+1)} = x_0 \to 2, x_1 \to 4, x_2 \to 16, x_3 \to 256, x_4 \to 65536, x_5 \to 4294967296,$$

$$x_6 \to 18446744073709551616, x_7 \to 340282366920938463463374607431768211456$$

Que es una función que tiende a infinito, mientras que si $x_0 = 0.5$:

$$f_{(x+1)} = x_0 \to 0.5, x_1 \to 0.25, x_2 \to 0.0625, x_3 \to 0.00390625, x_4$$

$$\to 0.0000152587890625, x_5 \to 2.328306436538696 \times 10^{-10}, x_6$$

$$\to 5.421010862427522 \times 10^{-20}, x_7 \to 2.938735877055719 \times 10^{-39}$$

Este ejemplo tiende a 0. También se puede crear una función que utilice más de un paso previo para su cálculo, como es el caso de la serie de Fibonacci[67]:

$$x_{(n+2)} = x_{(n+1)} + x_n$$

$$x_0 = 1$$

$$x_1 = 1$$

$$x_{n+1} = x_0 \to 1,\ x_1 \to 1, x_2 \to 2,\ x_3 \to 3,\ x_4 \to 5,\ x_5 \to 8,\ x_6 \to 13,\ x_7 \to 21$$

Ecuación 4

67 Que también tiende a infinito.

Una función iterada pertenece al conjunto de sistemas dinámicos discretos ya que genera datos que varían en el tiempo (dinámicos) a intervalos fijos (discretos). Entonces, para que un sistema dinámico se considere caótico debe reunir las siguientes condiciones (Feldman, 2012):

- La regla que produce la órbita del sistema dinámico debe ser determinista.
- Las órbitas son aperiódicas.
- Las órbitas están delimitadas.
- El sistema dinámico tiene una dependencia sensible de las condiciones iniciales.

Se han investigado gran cantidad de diferentes funciones que generan mapas caóticos[68]. Algunos de estos mapas son de dimensión 1 (es decir que poseen un solo dato de entrada *r* y a partir de allí generan una sola lista de salida), mientras que otros son de 2, 3 y más dimensiones. Aquí se exponen los mapas que se utilizan en la herramienta de composición creada para esta tesis junto con sus diagramas de bifurcaciones.

2.3.2 - Mapas de dimensión 1

Mapa logístico

Esta función (o mapa), fue introducida en 1838 por Pierre François Verhulst (1804 - 1849) como un modelo de crecimiento poblacional. Se define con la siguiente función iterada:

$$x_{n+1} = r x_n \left(1 - x_n\right)$$

Ecuación 5

68 Wikipedia enlista 139 de ellos https://en.wikipedia.org/wiki/List_of_chaotic_maps

Donde *r* es la tasa de crecimiento y *x* el valor inicial. Si se grafican 15 iteraciones de esta función, donde $r = 8$ y $x_0 = 0.1$, se observa que, a medida que *x* avanza, la órbita de la función tiende a 0; aplicado a un modelo de crecimiento de población significaría que la población desaparece (figura 39).

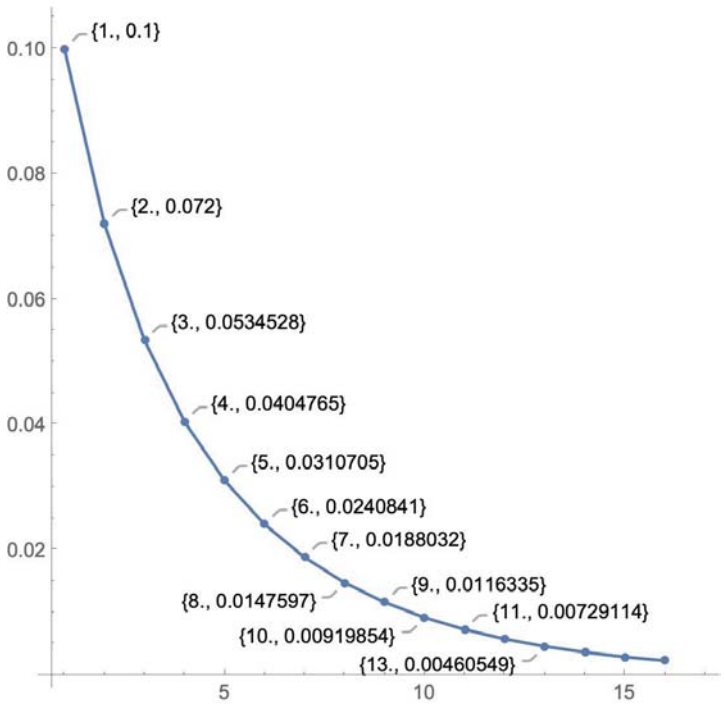


Figura 39: quince iteraciones de la función logística

Esto será así para diferentes valores de *r* < *1*. Sin embargo, a partir de *r* = 1 se produce una bifurcación. La curva ya no tiende a 0 y se estabiliza en un valor que crece a medida que *r* aumenta. Si *r* = *1.1*, la curva se estabiliza en 0.0909091 (figura 40).

Este comportamiento (la permanencia de la curva en un punto fijo de valor creciente a medida que *r* aumenta) se mantendrá hasta *1*< *r* < *3*. Por ejemplo, *r* = *2* tiende a un valor fijo en 0.5, *r* = *2.4* tiende a un valor aproximado de 0.6. Sin embargo, cuando se supera *r* = 3,

puede observarse que los puntos fijos tienden hacia dos valores diferentes (es decir que se repiten con un período igual a 2). En el video 2 se aprecia una animación con diferentes valores de *r* desde *1* hasta *3.45.*

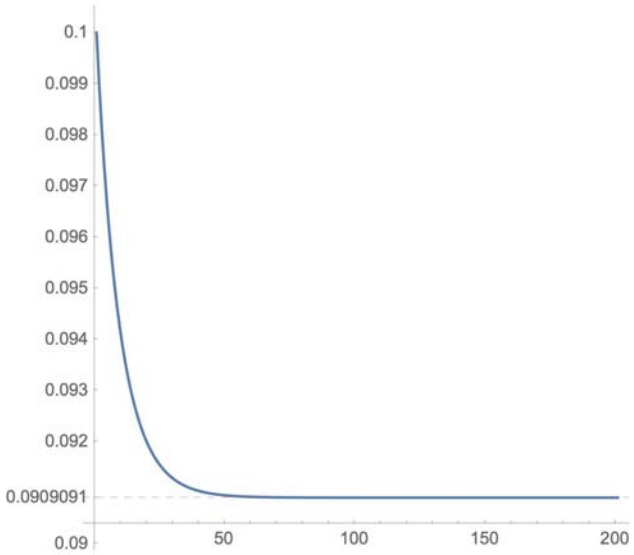
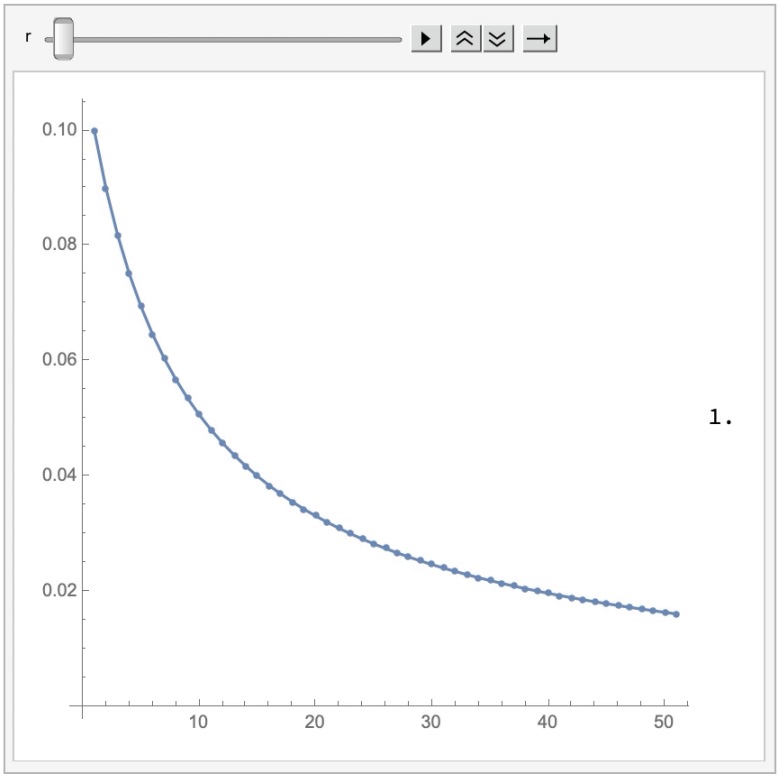


Figura 40: *r = 1.1*



Video 2: devolución de mapa logístico con *r* = 1 ~ 3.45

Estas bifurcaciones se siguen duplicando a medida que *3* < *r* < *4*. Aquí se suele utilizar un gráfico llamado *diagrama de bifurcaciones* que muestra este comportamiento de manera más clara (figura 41).

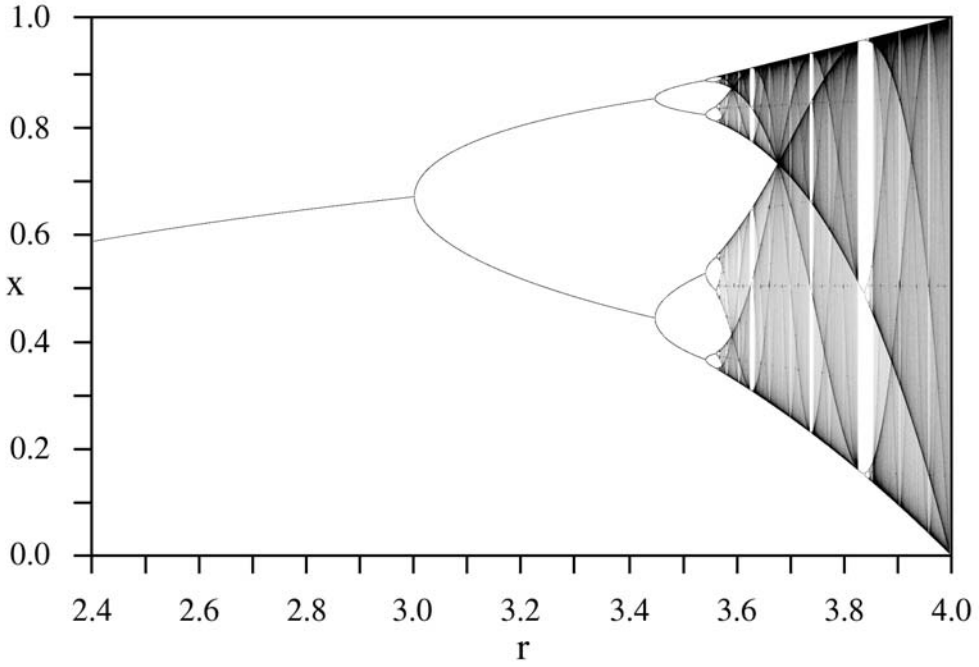


Figura 41: diagrama de bifurcaciones de un mapa logístico

Al analizar el video 2 se aprecia que allí algunas de las condiciones necesarias que menciona Feldman no se cumplen: si bien la regla que determina la órbita de la función está representada por la ecuación 5 y esta es determinista ya que, para los mismos datos de entrada, se obtienen los mismos de salida, las órbitas no son aperiódicas ya que, como se observa en dicha figura, esta presenta una periodicidad cada dos valores. Las órbitas sí están delimitadas porque no crecen al infinito sino que están acotadas a valores entre 0 y 1. Por último el sistema no tiene una dependencia a las condiciones iniciales, ya que para valores diferentes de condición inicial x_0 se obtienen los mismos resultados. En la figura 42 se observa un gráfico de una función logística con un valor *r = 3.56* y dos valores diferentes (pero muy próximos) para la condición inicial: 0.1 y 0.1001. Aquí es claro

que las dos órbitas son exactamente iguales (una está dibujada sobre la otra), esto significa que el sistema no es sensible a la condición inicial además de ser periódico. A partir de que *r* toma un valor aproximado de 3.56995 y hasta 4 el comportamiento se vuelve caótico.

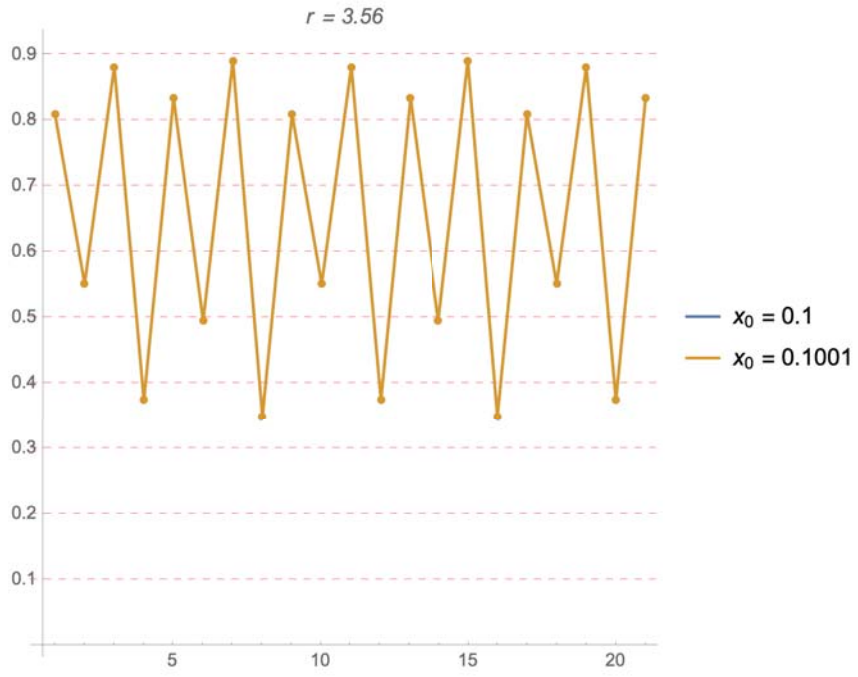


Figura 42: r = 3.56 y x_0 = 0.1 y 0.1001

En la figura 43, se observa un gráfico del mismo mapa logístico anterior, con las mismas condiciones iniciales, pero con r = 3.75. Se aprecia allí claramente que las cuatro condiciones expresadas por Feldman se cumplen.

Por otro lado es posible observar en el diagrama de bifurcaciones de la figura 41, que existen excepciones al comportamiento que se obtiene cuando *3.56995 < r < 4*. Aproximadamente cuando *r ≅ 3.82843* (figura 44) se ve lo que se denomina una *isla de estabilidad* o *ventana periódica* donde el sistema (durante algunos valores de *r*) se vuelve

a comportar de manera no caótica volviéndose a dividir en períodos de 2, 4 , 8, etc… hasta volverse caótico nuevamente.

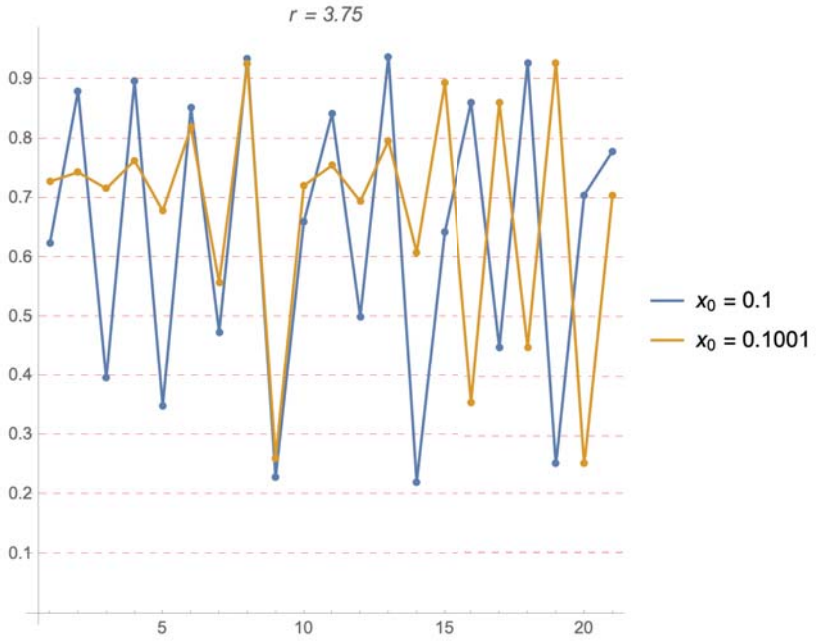


Figura 43: r = 3.75 y x_0 = 0.1 y 0.1001

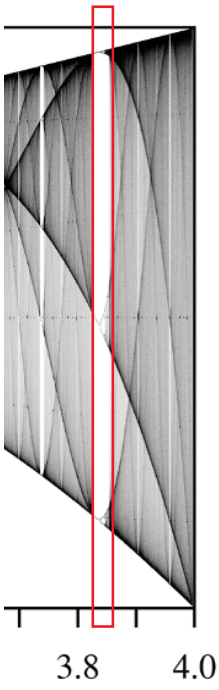
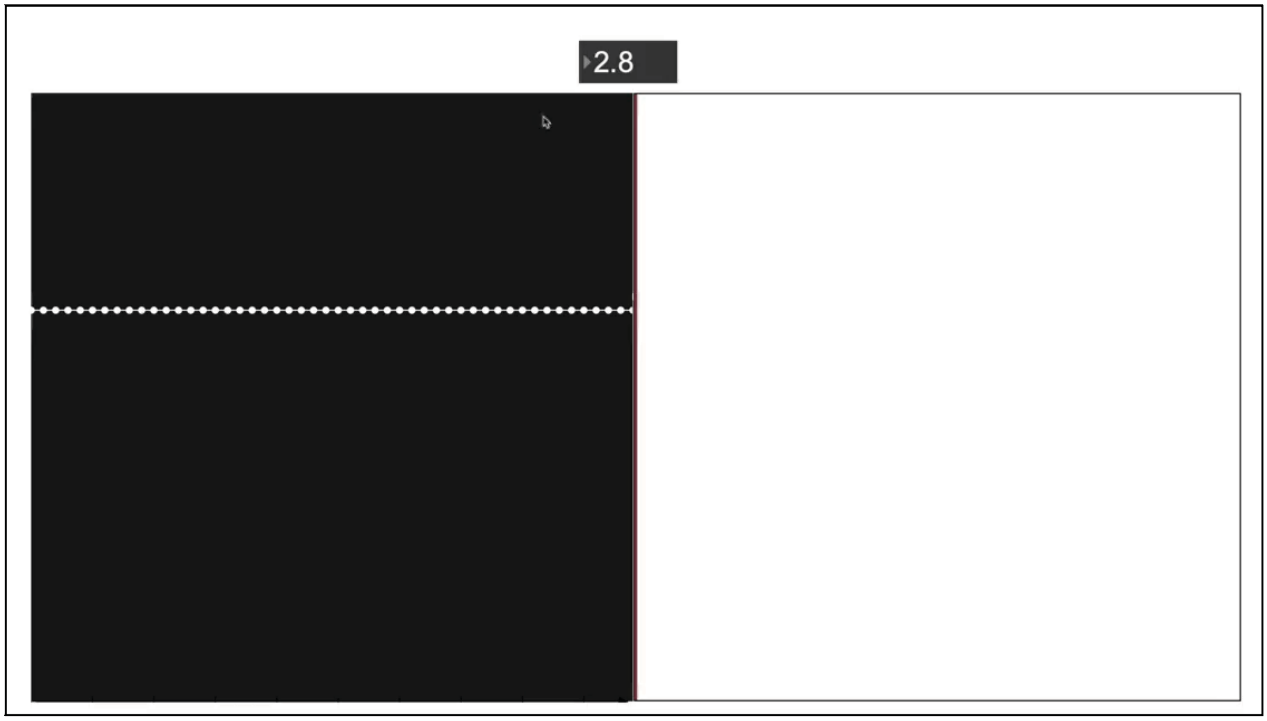


Figura 44: *isla de estabilidad*

Existen en realidad una infinita cantidad de ventanas periódicas entre $r \cong 3.56995$ y $r = 4$, sin embargo, la mayoría de ellas son tan estrechas que es poco probable encontrarlas.

En el video siguiente se observa con más claridad la representación gráfica y el diagrama de bifurcaciones de un mapa logístico. Se aprecia primero cómo se dibuja la imagen y luego en una segunda pasada más lenta se observa cómo los datos obtenidos se dividen en grupos de 2 y luego de 4 valores; a continuación se siguen ramificando aunque, alrededor de $r = 3.8$, aparece una *isla de estabilidad.*



Video 3: gráfico del mapa logístico y su diagrama de bifurcaciones

Congruencial lineal

Es un generador de números pseudoaleatorios de fácil implementación propuesto por D.H. Lehmer (1905 - 1991) en 1951 (L'Ecuyer, 2017). La ecuación para generarlo es la siguiente:

$$x_{n+1} = \left(a\,x_n + c\right) Mod(m)$$

Ecuación 6

Al utilizarse en lenguajes de programación como generador aleatorio, los parámetros *a*, *c* y *m* adquieren valores específicos: *a* = 1103515245, *c* = 12345, *m* = 2147483648 (es decir 2^{31}). Así se obtiene una secuencia pseudoaleatoria con un nivel de uniformidad aceptable para ciertos usos. En la figura 45 se muestra un histograma de un millón de recursiones:

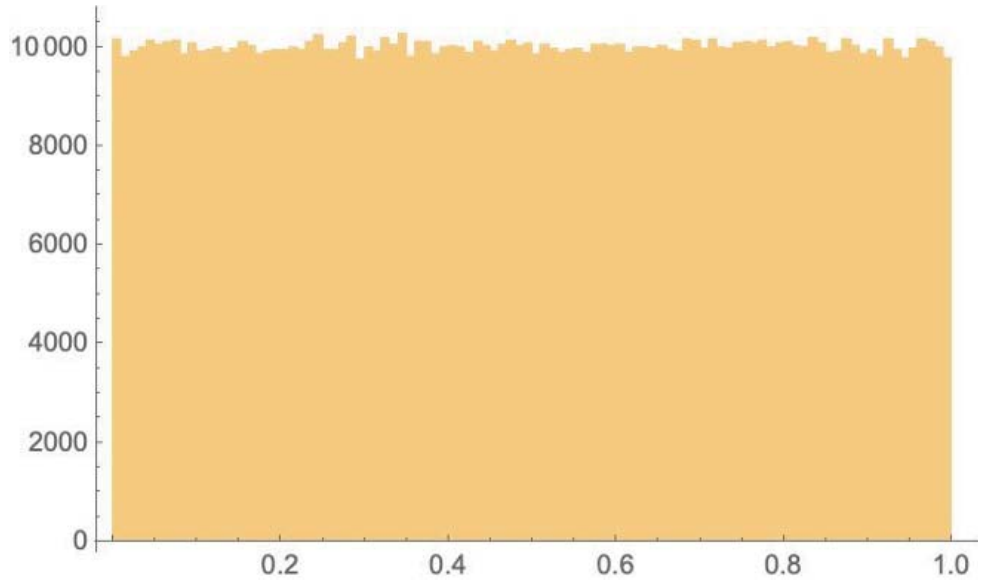


Figura 45: histograma de una función congruencial

Cuando a los parámetros antes mencionados se les asignan valores mas pequeños se obtiene una distribución que genera ciertos patrones interesantes para ser utilizados en un contexto musical. Por ejemplo, si se especifican los siguientes valores: *a* = 0.99, *c* = 2, *m* = 7 se obtienen una serie de valores como los que se observan en el gráfico de las figura 46.

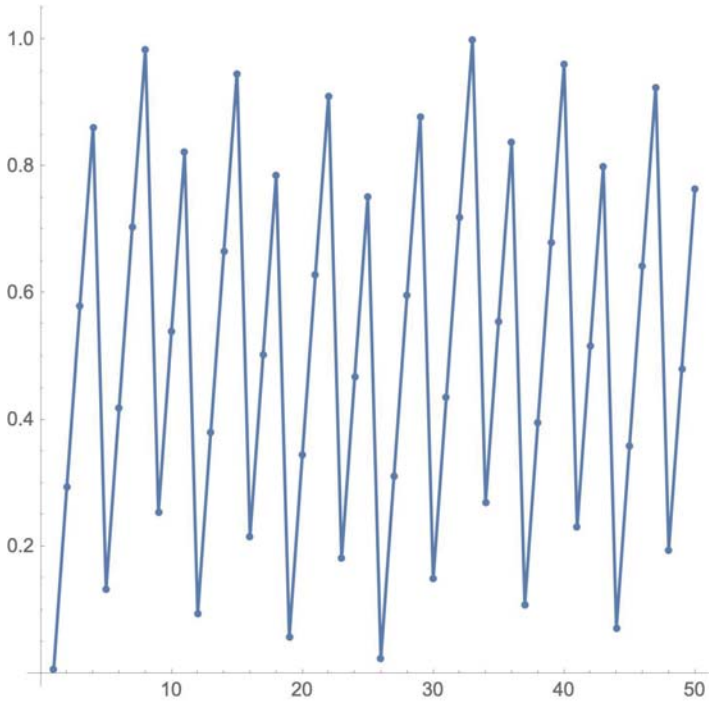


Figura 46: gráfico de mapa congruencial

A continuación un gráfico de bifurcaciones al variar el valor de *a* desde 0.1 a 1:

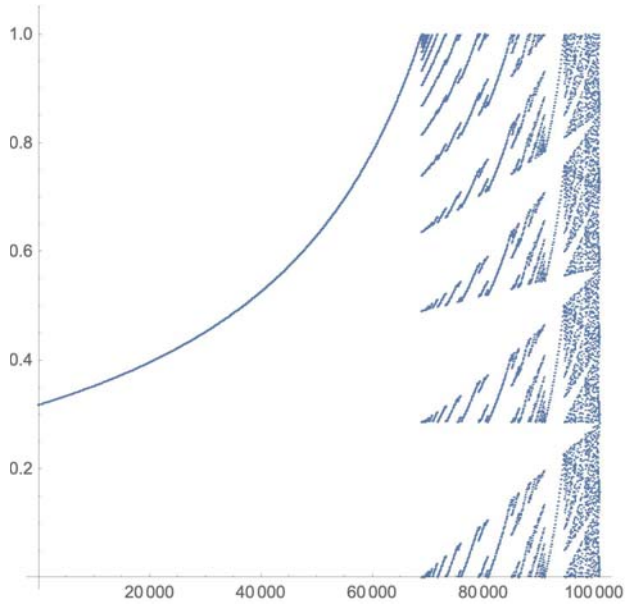


Figura 47: diagrama de bifurcaciones de un mapa congruencial

Sine Map

Esta función es similar al mapa logístico con la diferencia de que genera un diagrama de bifurcación que es simétrico con respecto al eje *x*. En la librería de objetos *Chaos* de *OpenMusic*[69] y en los objetos *a-chaos lib* para *Max* de André Sier[70] se llama a este mapa *chaos stein*[71]. La ecuación de esta función es:

$$x_{n+1} = rsin\left(\pi x_n\right)$$

Ecuación 7

En el caso de este mapa, como se observa en la ecuación 7, la variable *r* escala los valores resultantes de $sin\left(\pi x_n\right)$. Por lo tanto variar dicho dato solo incrementa la distancia entre los puntos generados. Si no se modifica *r,* se obtiene*:*

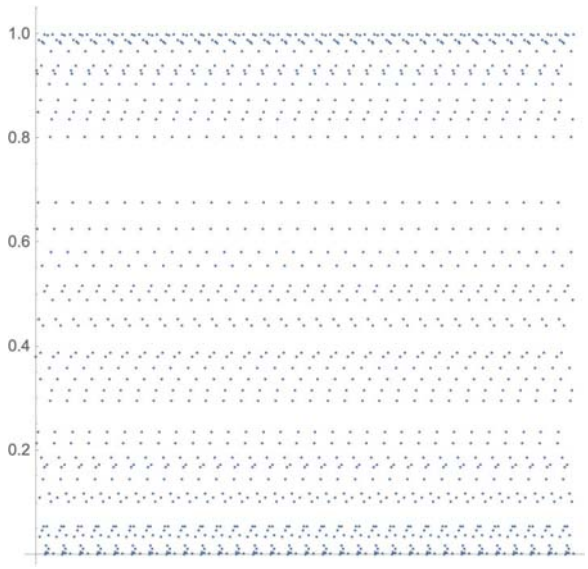


Figura 48: Sine Map donde r siempre es igual a 1

69 https://openmusic-project.github.io

70 https://andre-sier.com

71 Presumiblemente por Paul R. Stein (Stein, 1987)

Como se observa en la figura 48, los valores se mantienen estables (con pequeñas variaciones) durante, en este caso, 30 iteraciones. Sin embargo si se varía el valor de *r* en el tiempo (figura 49), se aprecia cómo la distancia entre los puntos se expande a medida que *r* crece.

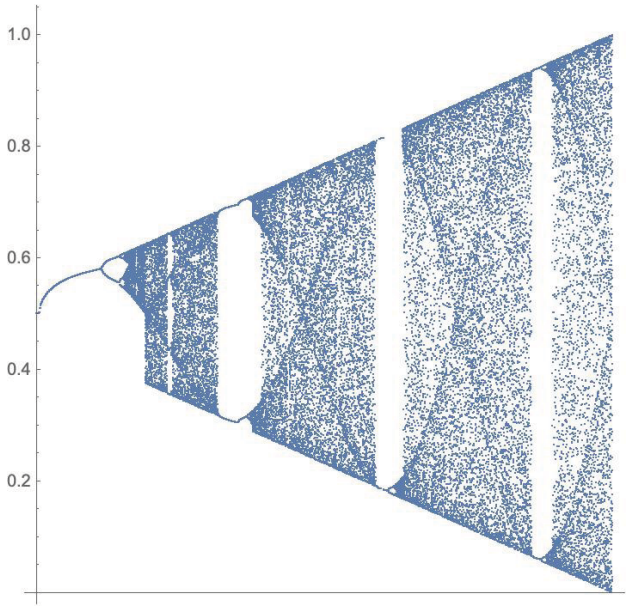


Figura 49: Sine Map donde *r* va creciendo a lo largo del tiempo

CUSP Map

La ecuación para obtener este mapa es[72]:

$$x_{n+1} = a - b\sqrt{|x_n|}$$

Ecuación 8

72 Weinstein, Eric W. "Cusp map." From MathWorld--A Wolfram Web. Resource. https://mathworld.wolfram.com/CuspMap.html

Su diagrama de bifurcaciones:

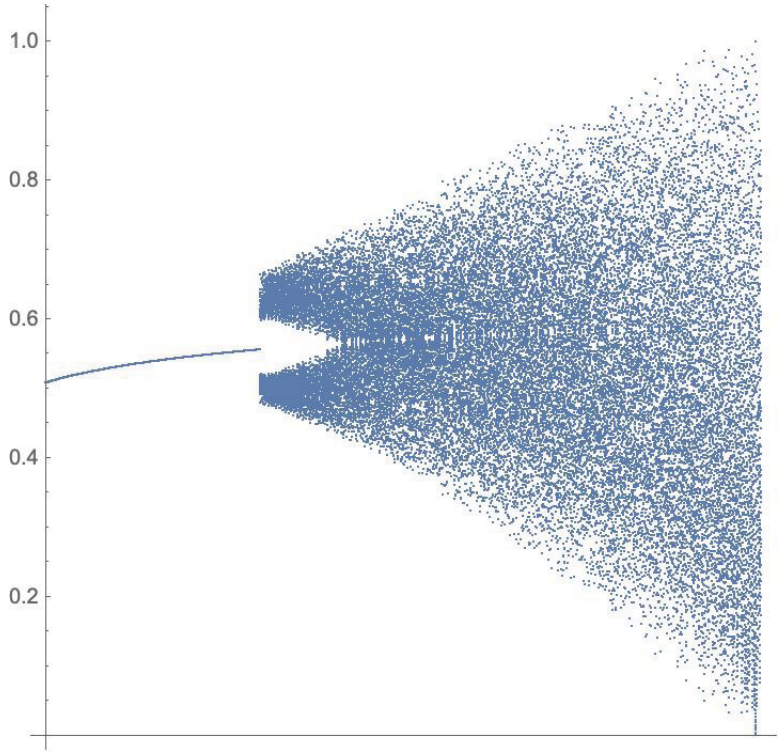


Figura 50: diagrama de bifurcaciones del mapa CUSP

Cuadrático

Una función cuadrática o polinomio de grado 2, posee una o más variables en la que el término de grado más alto es de segundo grado.

$$x_{n+1} = {x_n}^2 + c$$

Ecuación 9

Su diagrama de bifurcaciones:

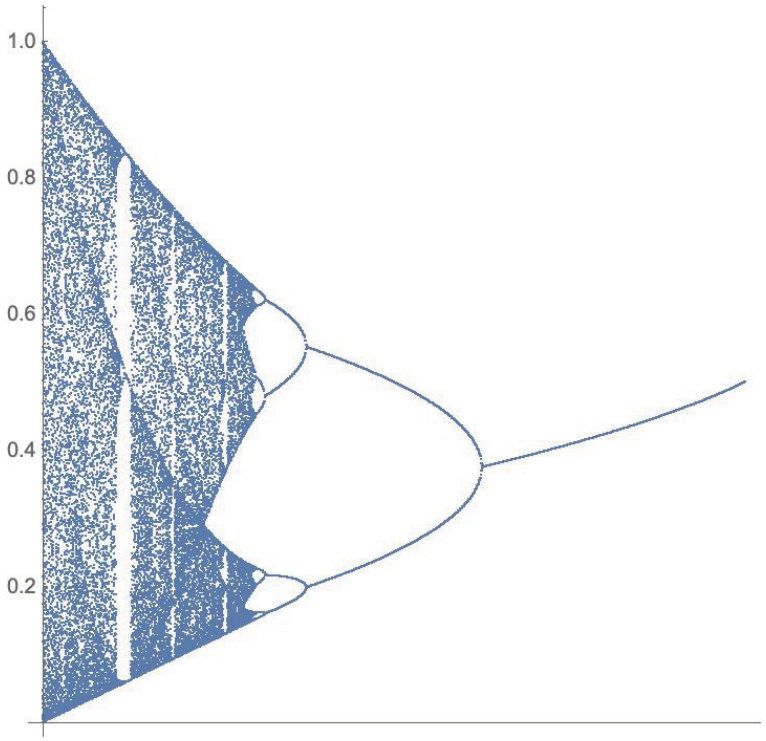


Figura 51: diagrama de bifurcaciones de mapa cuadrático

Gauss Map

Este mapa se produce por la iteración de una función Gauss. Su diagrama de bifurcación se asemeja a la figura de un ratón por lo que a veces se lo llama *Mouse Map*.

$$x_{n+1} = e^{-a\ x_n{}^2} + b$$

Ecuación 10

Su diagrama de bifurcaciones:

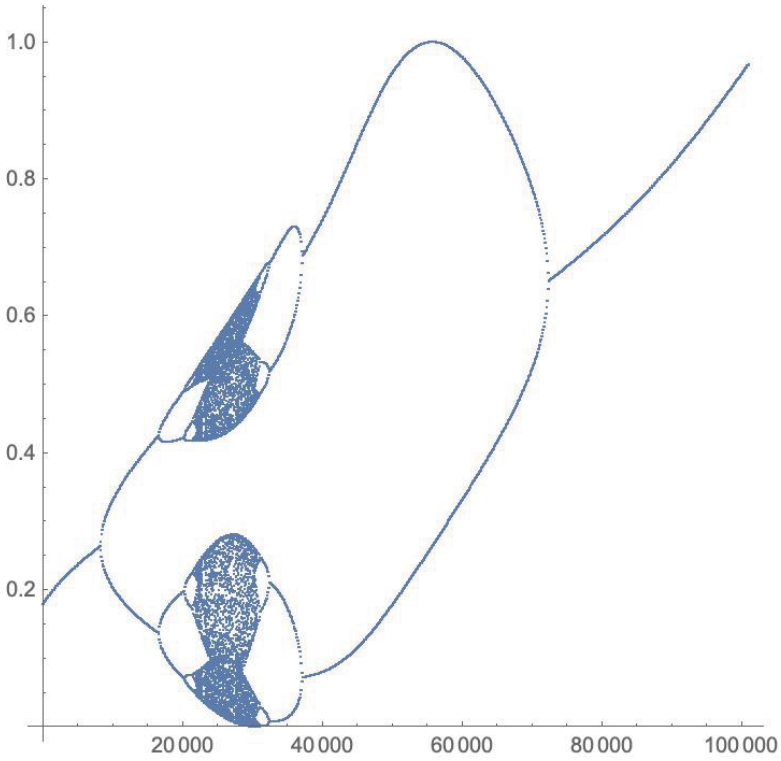


Figura 52: diagrama de bifurcaciones de mapa cuadrático

2.3.3 - Mapas de dimensión 2

Estos mapas poseen dos órbitas de salida. En dicho escenario se obtienen dos secuencias discretas dinámicas a partir de dos valores de entrada. Esto, sin embargo, no es lo mismo que implementar dos funciones separadas, sino que es un proceso con una entrada y una salida múltiple en la cual las entradas están asociadas. Un ejemplo de esto sería una función como la siguiente:

$$x_{n+1} = {x_n}^2 + y_n$$

$$y_{n+1} = 2\ y_n$$

Ecuación 11

Aquí se aprecia esta ecuación implementada utilizando software Mathematica[73]:

```
TableForm[
 RecurrenceTable[{{x[n + 1] == x[n]^2 + y[n], x[0] == .5},
   {y[n + 1] == 2 y[n] , y[0] == .5}}, {x, y}, {n, 0, 5}],
 TableHeadings → {None, {"xn", "yn"}}]
```

Out[1265]//TableForm=

x_n	y_n
0.5	0.5
0.75	1.
1.5625	2.
4.44141	4.
23.7261	8.
570.927	16.

Figura 53: implementación de la ecuación 11 utilizando Mathematica

Se observa claramente que $x_0 = 0.5, y_0 = 0.5$, ya que esas son las condiciones iniciales. Luego, para x_1, y_1 se obtiene $x_1 = 0.75$, ya que 0.5 (que es el último valor de *x*) elevado al cuadrado da 0.25, y al sumarse 0.5 (que es el último valor de *y*) da 0.75. El valor de *y* se va duplicando ya que $y_{n+1} = 2\ y_n$

Henon

Este mapa caótico adopta su forma clásica cuando *a = 1.4* y *b = 0.3*. Dicha forma es la que se observa en la figura 54, donde se grafican las órbitas *x*/*y* en el plano cartesiano.

$$x_{n+1} = 1 - a\ x_n^2 + y_n$$

$$y_n = b\ x_n$$

Ecuación 12

[73] https://www.wolfram.com/mathematica/

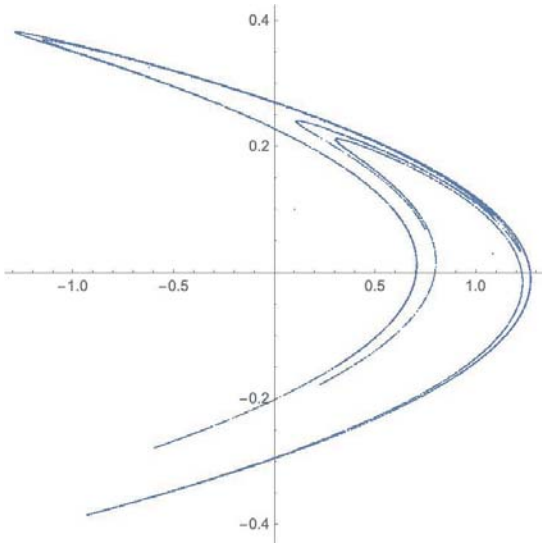


Figura 54: gráfico de un mapa de Henon en un plano cartesiano

Al apreciar un gráfico de 20 valores de las órbitas *x* e *y* (tomado los valores que se encuentran entre las iteraciones 1000 y 1020[74]) se ve que ambas órbitas poseen un perfil similar, aunque corrido en su fase por una muestra y con diferente amplitud (figura 55).

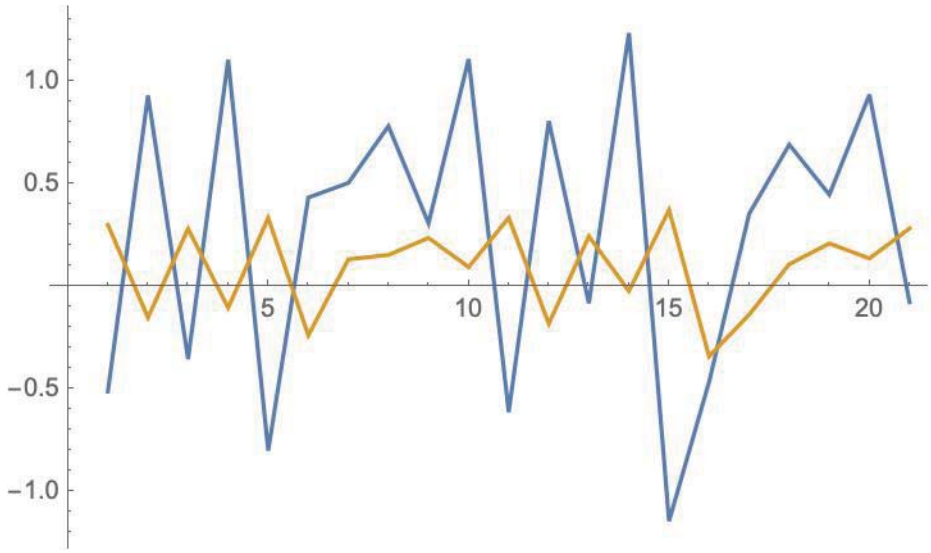
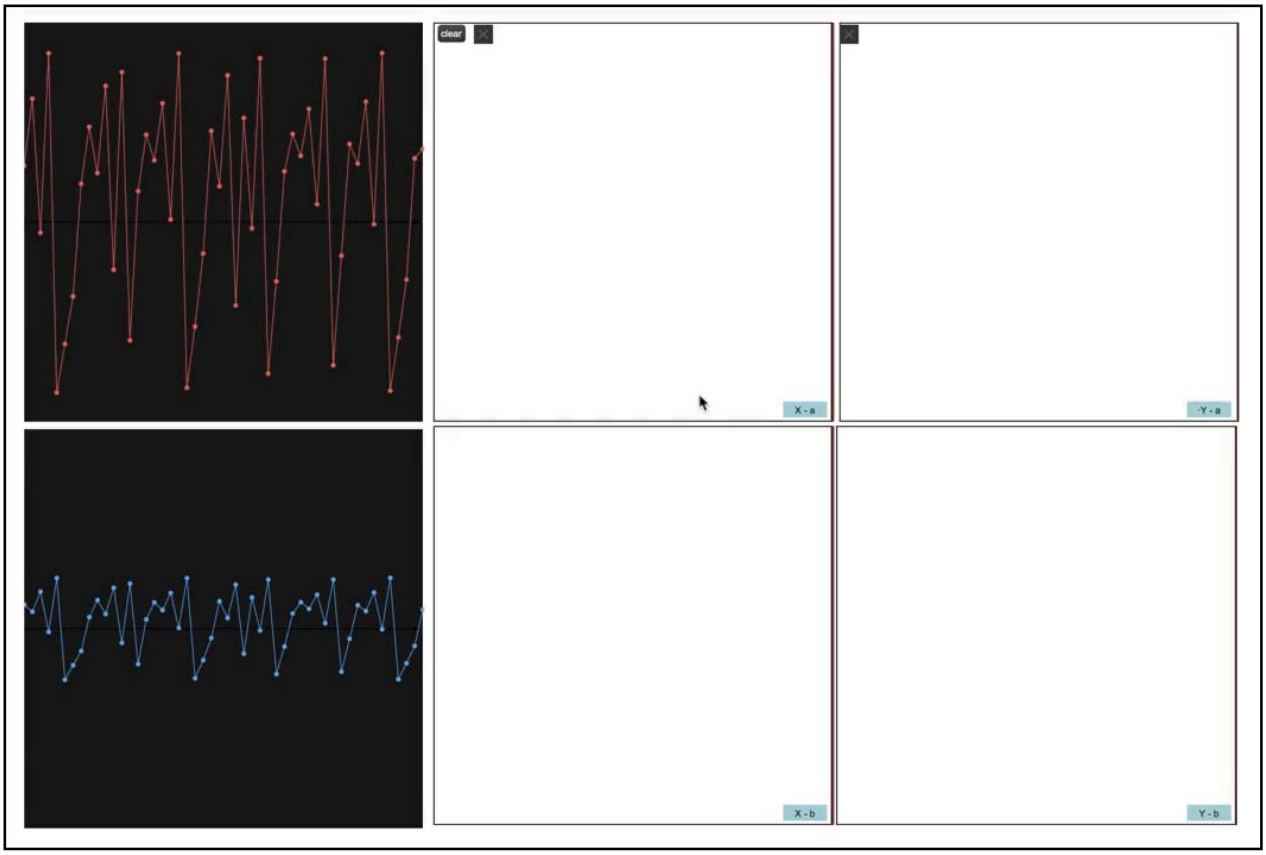


Figura 55: órbitas de un mapa Henon

[74] De un total de 10,000 iteraciones.

En el video 4 se grafican los diagramas de bifurcaciones correspondientes a 140 órbitas de *x* (de un tamaño de 100 iteraciones cada una) en la que el valor de *a* va desde 0 a 1.4 en 100 pasos (diagrama de bifurcaciones central arriba), y lo mismo para el eje *y* (diagrama de bifurcaciones central abajo). Luego se observa lo mismo para el parámetro *b* en ambos ejes (diagramas de bifurcaciones de la derecha). Por último se observan los gráficos de cada eje en la parte izquierda.



Video 4: gráficos de los ejes x/y de un mapa Henon y su diagrama de bifurcaciones

Como se aprecia, los diagramas de bifurcaciones de los ejes *x* e *y,* cuando se modifican los parámetros de *a,* son muy similares (solo varían en escala); sin embargo la variación del parámetro *b* provoca resultados más disímiles para los ejes *x* e *y.*

Henon Phase

Esta ecuación está tomada de la colección de atractores extraños creada por Paul Bourke publicada en su web: http://paulbourke.net/fractals/

$$x_{n+1} = x_n cos\,(a) - \left(y_n - {x_n}^2\right) sin\,(a)$$

$$y_{n+1} = x_n sin\,(a) + \left(y_n - {x_n}^2\right) cos\,(a)$$

Ecuación 13

Algunas de las figuras que forma el gráfico de sus órbitas en el eje cartesiano son las siguientes:

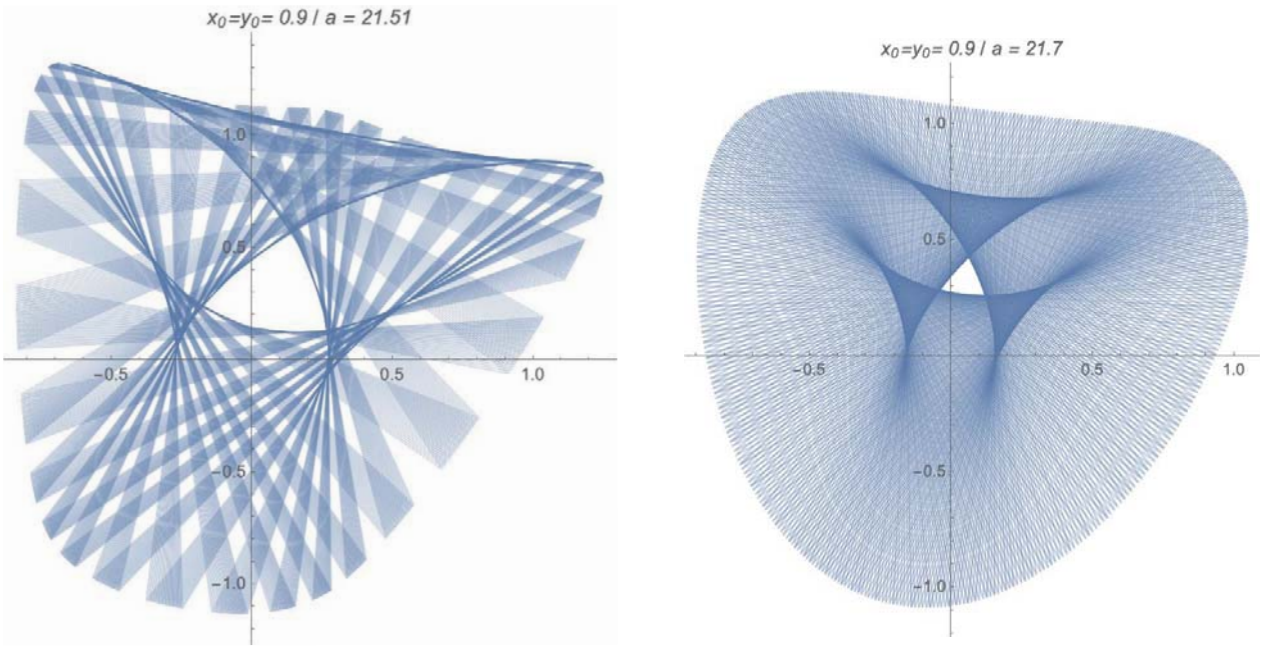


Figura 56: gráfico de las órbitas de un mapa Henon phase

El gráfico de sus órbitas independientes:

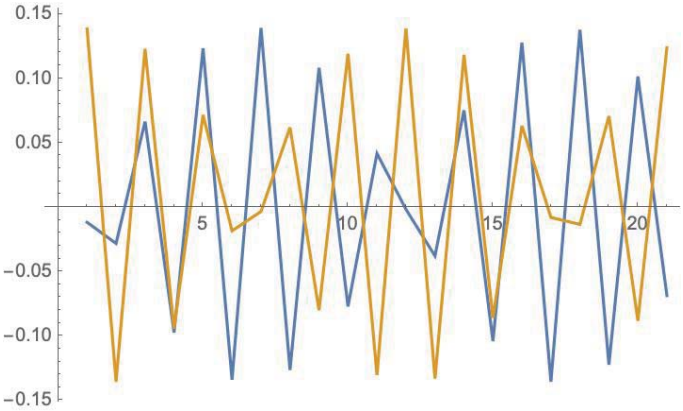
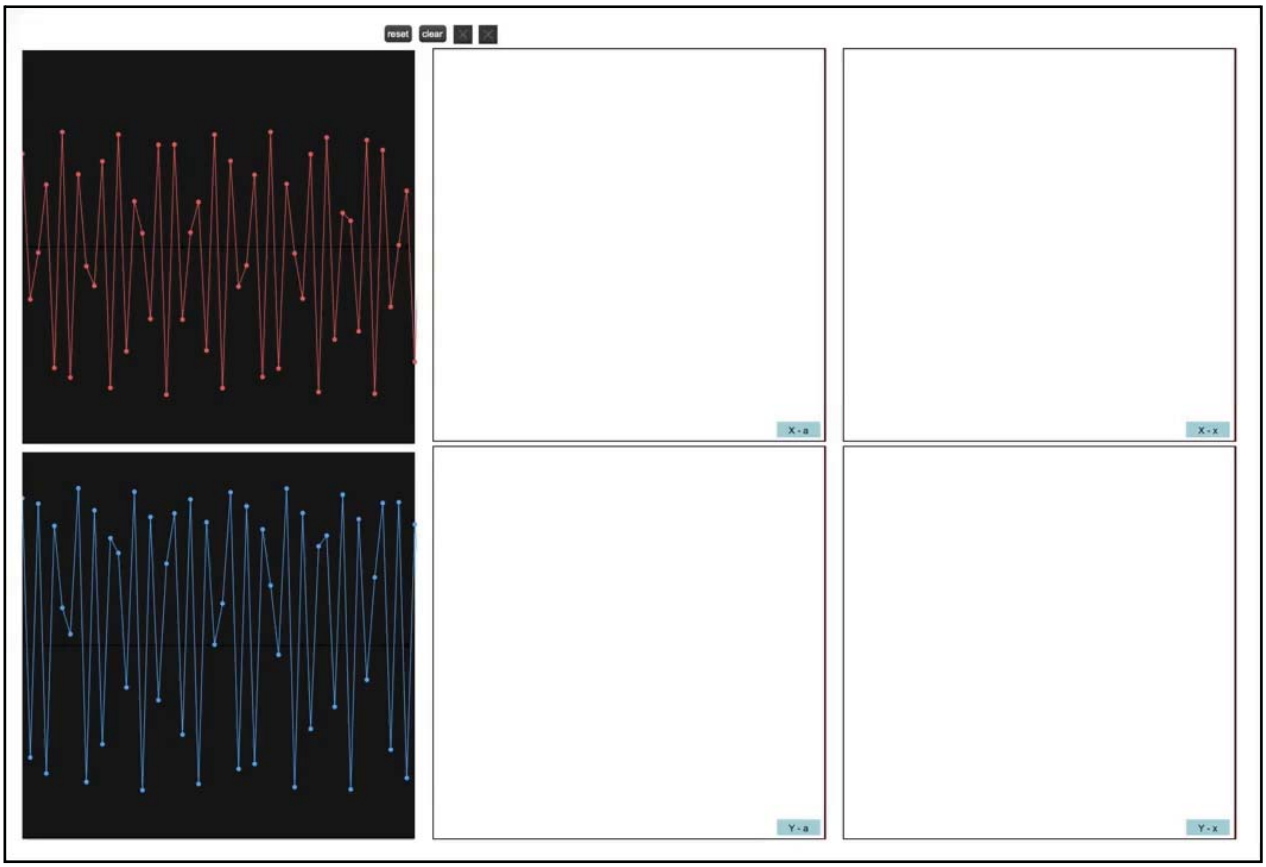


Figura 57: gráfico de las órbitas

En el siguiente video se observan los diagramas de las bifurcaciones y los gráficos de cada órbita. Como se ve, en este caso las órbitas poseen comportamientos más diferenciados que en el caso de Henon.



Video 5: gráficos de los ejes x/y de un mapa Henon Phase y sus bifurcaciones

Clifford

Este mapa es uno de los más conocidos dentro del mundo de las generaciones de formas a partir de funciones recursivas ya que pueden crearse un sinnúmero de imágenes muy interesantes. Se atribuye su descubrimiento al biofísico molecular estadounidense Clifford Alan Pickover(1957) [75] .

Un buen ejemplo de las múltiples posibilidades que este mapa genera a nivel gráfico se encuentra en los cincuenta y cinco atractores de Clifford generados por Danielle Romanoff[76] que se ven en la siguiente figura:

Figura 58: diferentes gráficos generados a partir del mapa Clifford

Su fórmula se define así:

75 http://www.pickover.com

76 https://towardsdatascience.com/from-fractals-to-attractors-b0e4ce9dcc71

$$x_{n+1} = sin\ a y_n + \ c\ cos\ a x_n$$

$$y_{n+1} = sin\ b x_n + \ d\ cos\ b y_n$$

Ecuación 14

El gráfico siguiente, generado en el software Mathematica, implementa los siguientes parámetros: a = 1.7, b = 1.7, c = 0.2 y d = 1.2. Con $x_0 = 0.1$, $y_0 = 0.1$

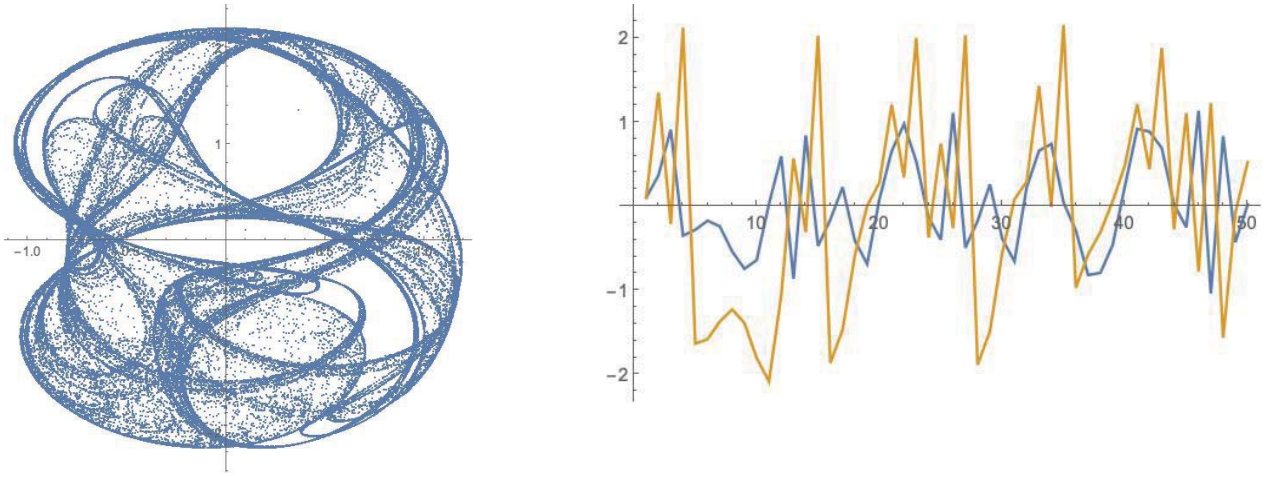


Figura 59: gráfico del mapa Clifford

En cuanto a su gráfico de bifurcaciones (figura 60), es posible obtener muchas variaciones de acuerdo al parámetro que se modifique en el tiempo. Aquí se especifican cuatro parámetros posibles (*a*, *b*, *c* y *d*). Se observa en la figura 60 las órbitas *x* e *y* al variar el parámetro *d*, luego el *a* y por último el *b*.

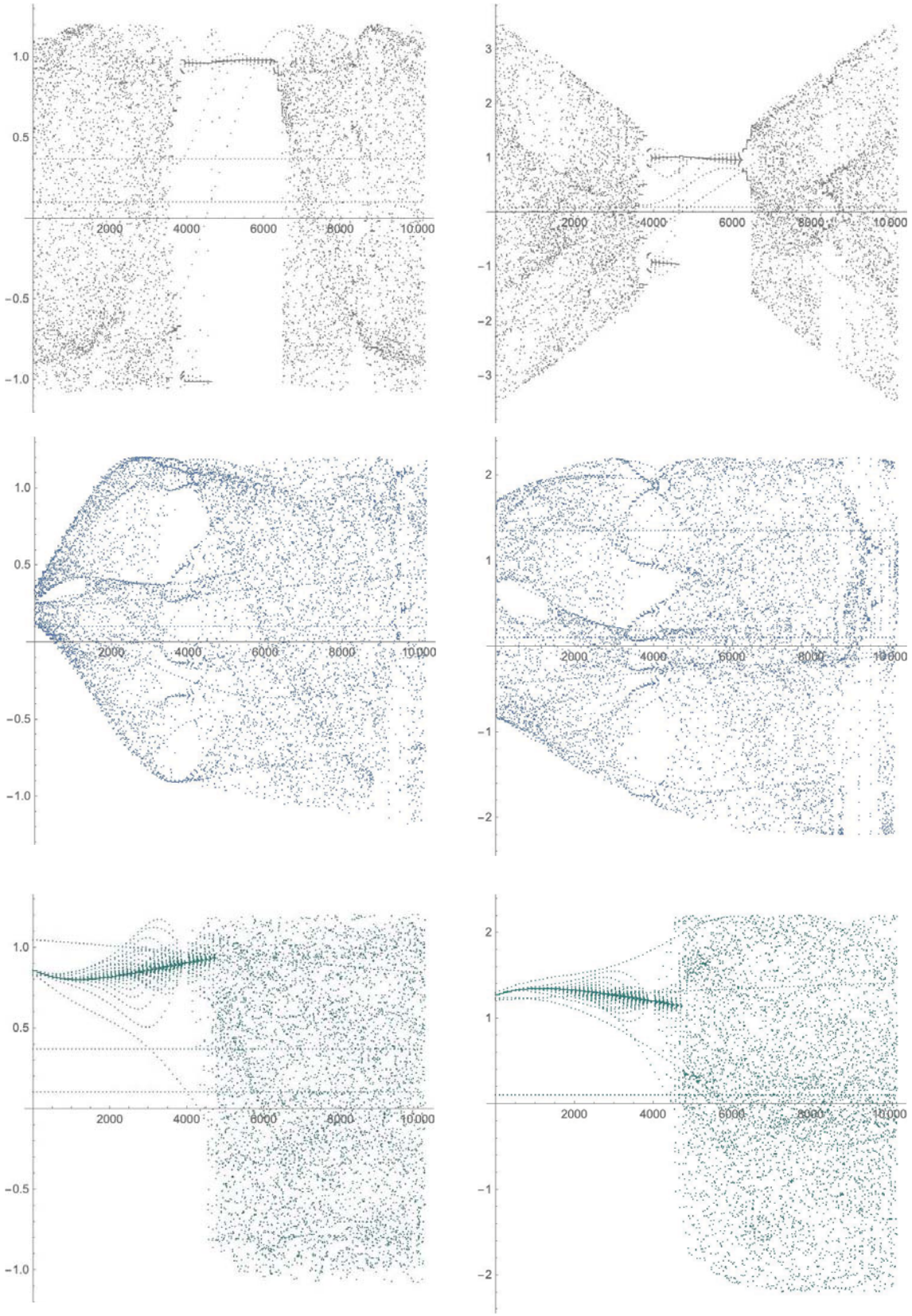


Figura 60: diagrama de bifurcaciones de Clifford

Jong

Este mapa es similar al anterior, solo cambia el signo + por – y no utiliza *a, c* y *d* para escalar los valores de los cosenos sino para su propio cálculo. Pueden observarse figuras muy interesantes generadas por Paul Bourke en su web[77]. Su nombre se debe a Peter de Jong (1945) citado por A. K. Dewdney en su artículo de Scientific American (Dewdney, 1987). Su fórmula:

$$x_{n+1} = sin\ a y_n - \ cos\ b x_n$$

$$y_{n+1} = sin\ c x_n - \ cos\ d y_n$$

Ecuación 15

A continuación se observa un gráfico de las órbitas *x* e *y* del mapa de Jong :

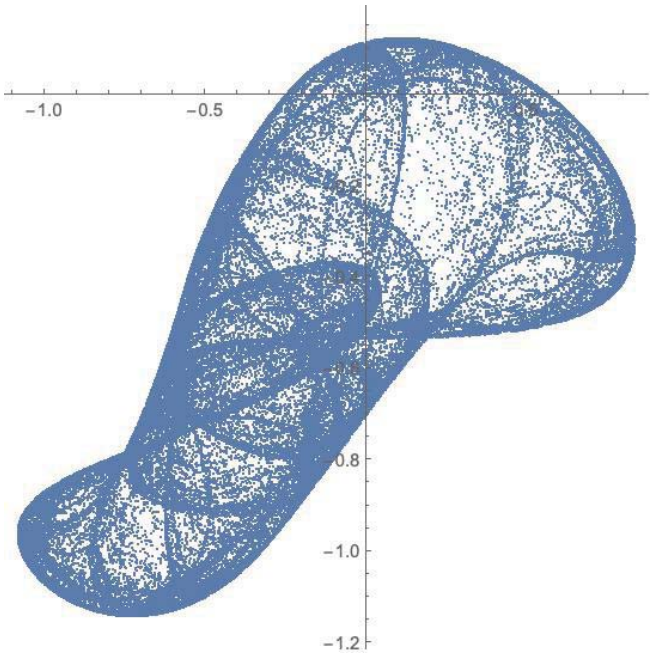


Figura 61: gráfico de las órbitas x/y de un mapa Jong

77 http://paulbourke.net/fractals/peterdejong/

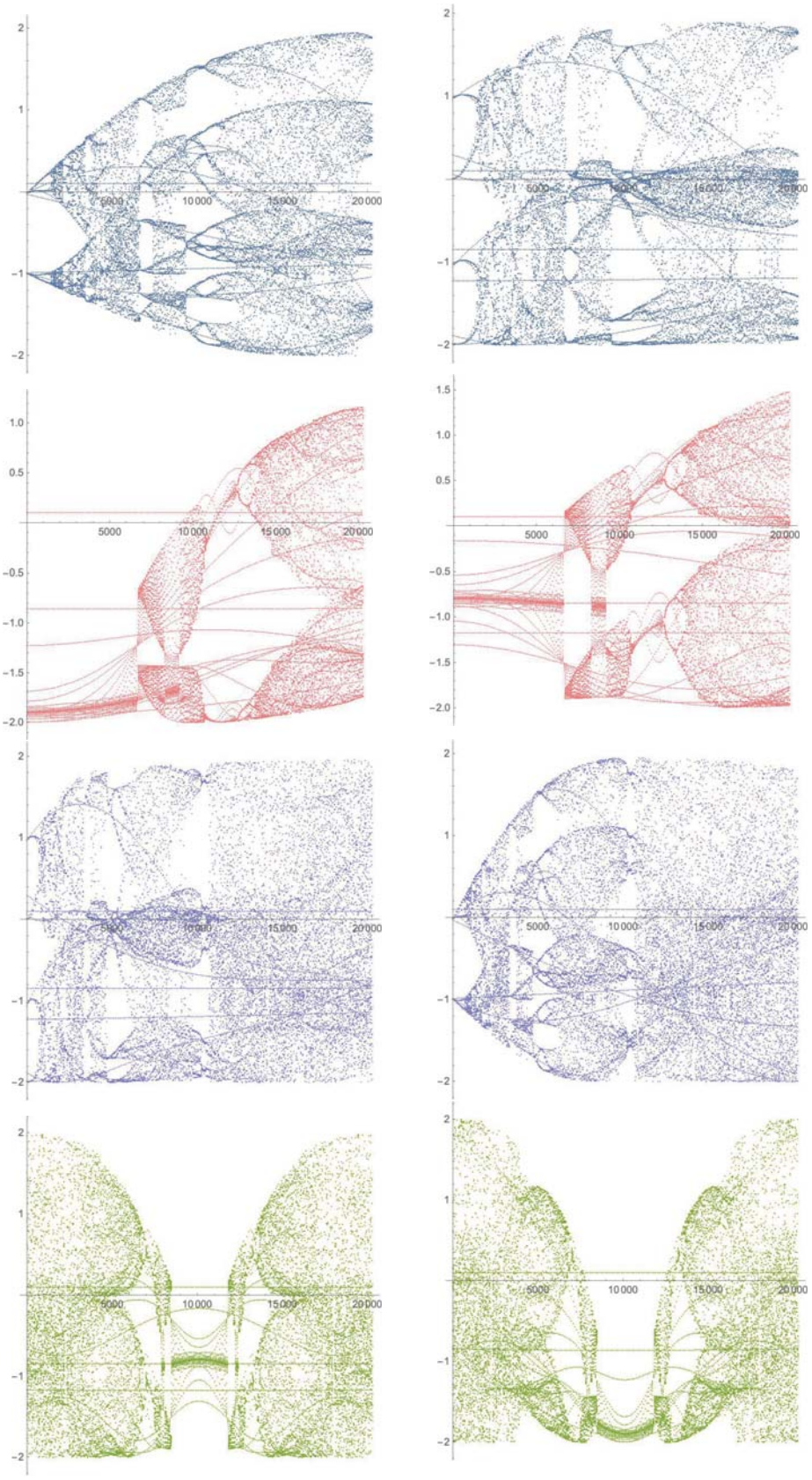


Figura 62: diagrama de bifurcaciones de Jong

Aquí también el gráfico de bifurcaciones (figura 62) puede adoptar muchas formas dependiendo de cuál o cuáles parámetros se varíen en el tiempo. Esto puede aportar una fuente muy rica de variaciones posibles a la hora de convertir estos valores en parámetros musicales. En la figura 62 se aprecian los gráficos generados al modificar los parámetros *a, b, c, d* respectivamente.

Gingerbread man

El nombre de este sistema viene del dibujo que forman sus órbitas al graficarlas en el plano cartesiano ya que esta se asemeja a un “hombre de jengibre” que es un conocido y muy popular tipo de galleta en USA. En la figura 63 se observa el gráfico generado en Mathematica y una imagen de la galleta mencionada.

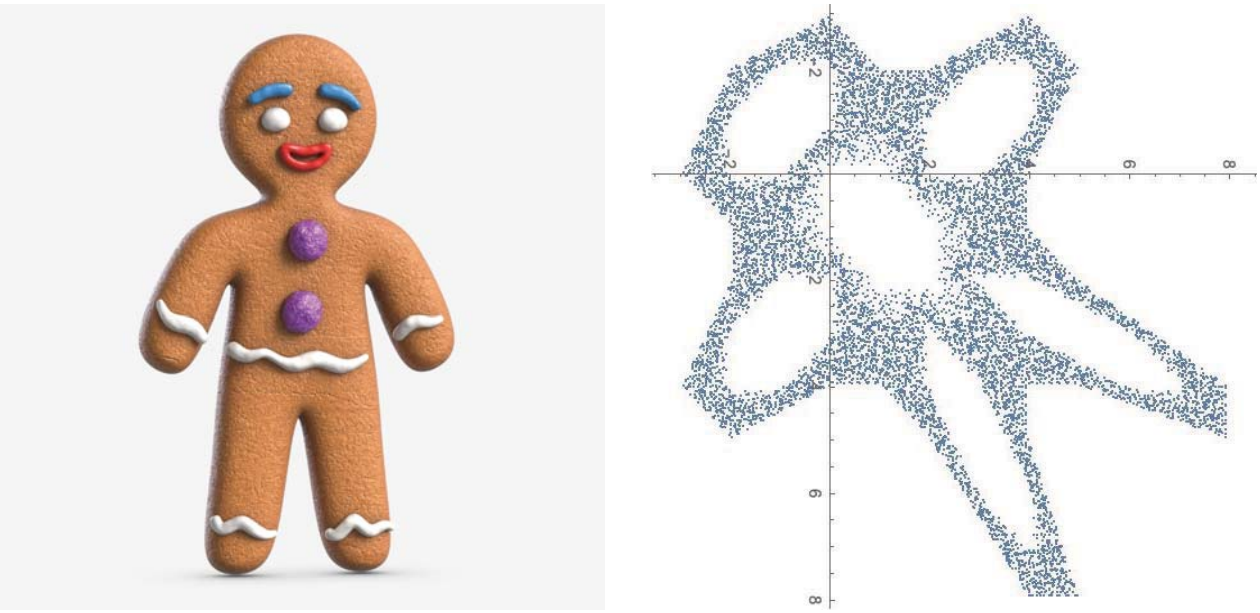


Figura 63: galleta llamada “hombre de jengibre” y gráfico

Su fórmula es:

$$x_{n+1} = 1 - y_n + |x_n|$$

$$y_{n+1} = x_n$$

Ecuación 16

Aquí los únicos parámetros que se pueden modificar en el tiempo son los valores iniciales $(x_0\ ,\ y_0)$. Las órbitas generadas para *x* e *y* son iguales por lo que a continuación se muestra un gráfico de distribuciones de la órbita *x* cuando avanza desde $x_0 = 0.1$ hasta $x_{100} = 10$

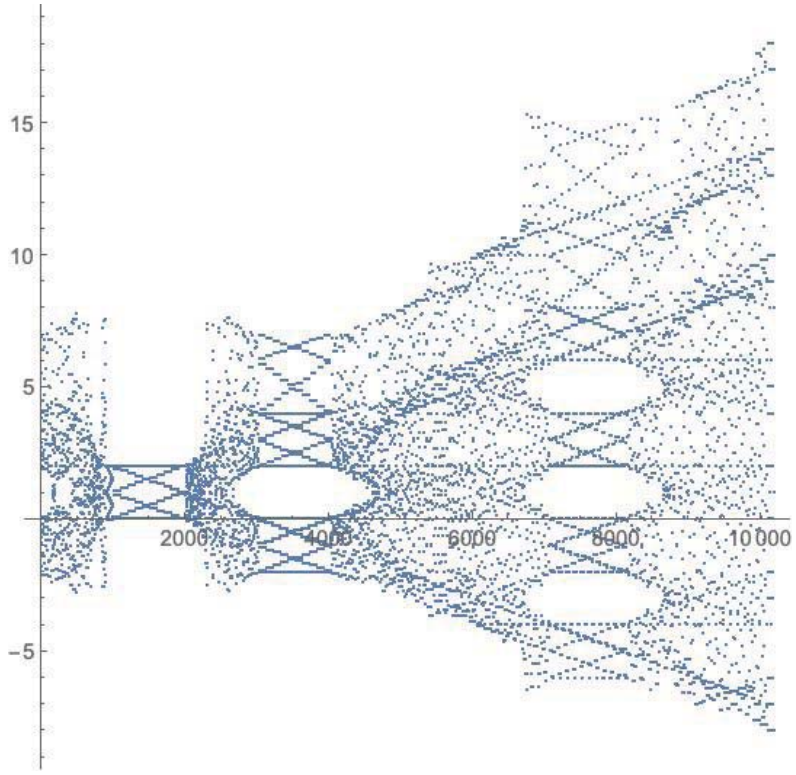


Figura 64: diagrama de bifurcaciones de Gingerbread man

Burning ship

Esta ecuación genera una figura de tipo fractal que se asemeja a un barco quemado y de allí su nombre (ver figura 65). Fue propuesto por Michael Michelitsch (1963) y Otto E. Rössler (1940) (Michelitsch & Rössler, 1992). Paul Bourke en su web muestra la siguiente figura[78]:

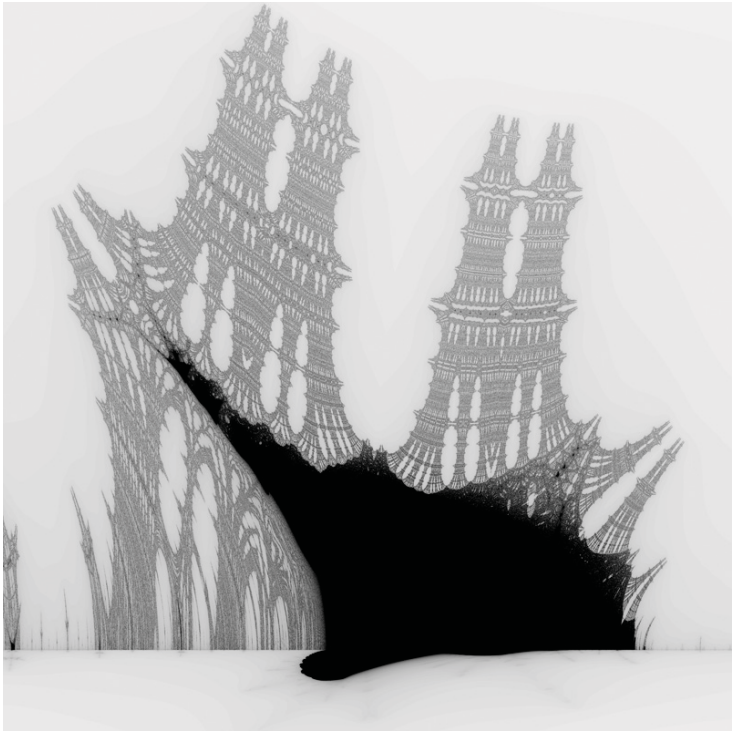


Figura 65: gráfico de un mapa de tipo Burning ship.

Su fórmula es:

$$x_{n+1} = {x_n}^2 - {y_n}^2 - c_x$$

$$y_{n+1} = 2 \left| x_n \; y_n \right| - c_y$$

Ecuación 17

78 Bourke explica la manera de generar dicha imagen: "El nivel de gris asignado a cada punto (cx, cy) está determinado por la rapidez con que la serie tiende al infinito. Por ejemplo, los puntos que se muestran en negro en el centro de la forma nunca divergen al infinito, sino que oscilan o convergen en un valor simple. Los puntos que se muestran en gris claro divergen muy rápidamente. Esta es exactamente la misma técnica utilizada para crear la imagen de Mandelbrot. En todos los casos, el punto de partida (x0, y0) es (0,0).

A continuación se observa el gráfico de sus órbitas en el plano cartesiano, después de cien mil iteraciones, con $c_x = 0$ y $c_y = 0.9$ siendo los puntos de inicio $x_0 = y_0 = 0$:

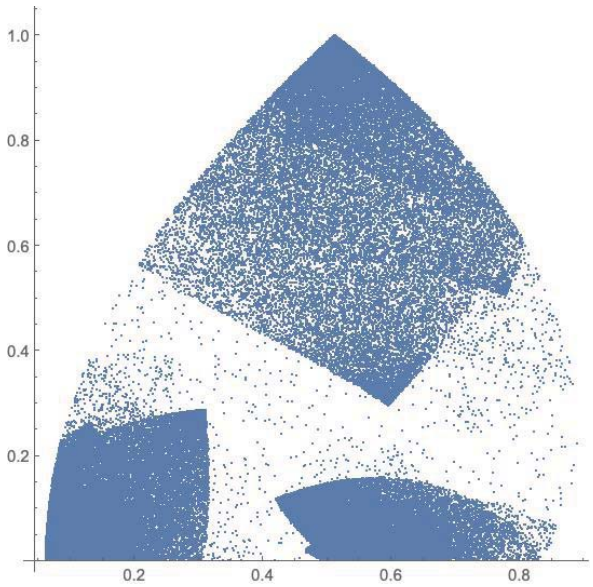


Figura 66: gráfico de un mapa Burning Ship

Se observa en la figura 67, el gráfico de las bifurcaciones para $x_0 = y_0 = 0$, $c_y = 0.2$ y c_x evolucionando desde 0.1 a 0.9 en 100 pasos.

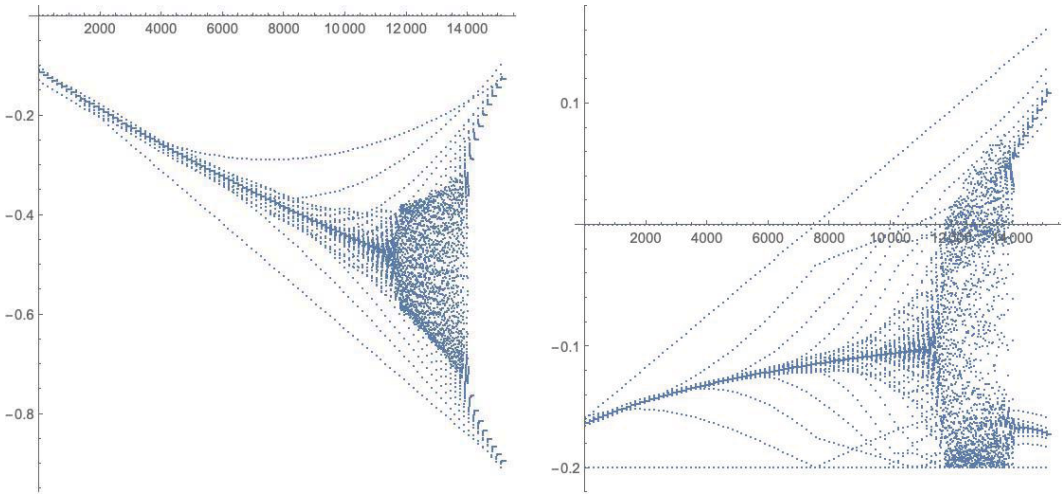


Figura 67: diagrama de bifurcaciones de Burning Ship

Ikeda

El mapa original fue propuesto por el físico japonés Kensuke Ikeda como un modelo de luz que circula por un resonador óptico no lineal[79]. Este modelo fue publicado en un artículo del *Optics Communication Journal* (Ikeda, 1979). Su fórmula es:

$$x_{n+1} = 1 + a\left(x_n cos\left(c - \frac{b}{1 + {x_n}^2 + {y_n}^2}\right) - y_n sin\left(c - \frac{b}{1 + {x_n}^2 + {y_n}^2}\right)\right)$$

$$y_{n+1} = a\left(x_n sin\left(c - \frac{b}{1 + {x_n}^2 + {y_n}^2}\right) + y_n cos\left(c - \frac{b}{1 + {x_n}^2 + {y_n}^2}\right)\right)$$

Ecuación 18

El gráfico de sus órbitas en el plano cartesiano, después de cien mil iteraciones, con a = 0.8, b = 9 y c = 0.5, siendo los puntos de inicio $x_0 = y_0 = 0$, se observa a continuación:

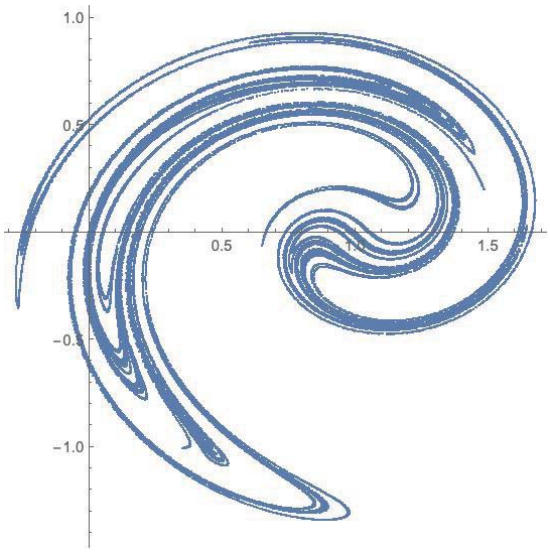


Figura 68: gráfico de las órbitas de un mapa Ikeda

[79] https://en.wikipedia.org/wiki/Ikeda_map

En la figura 69 se aprecian los diagramas de bifurcaciones y los histogramas de este mapa cuando se varían *a*, *b* y *c* respectivamente.

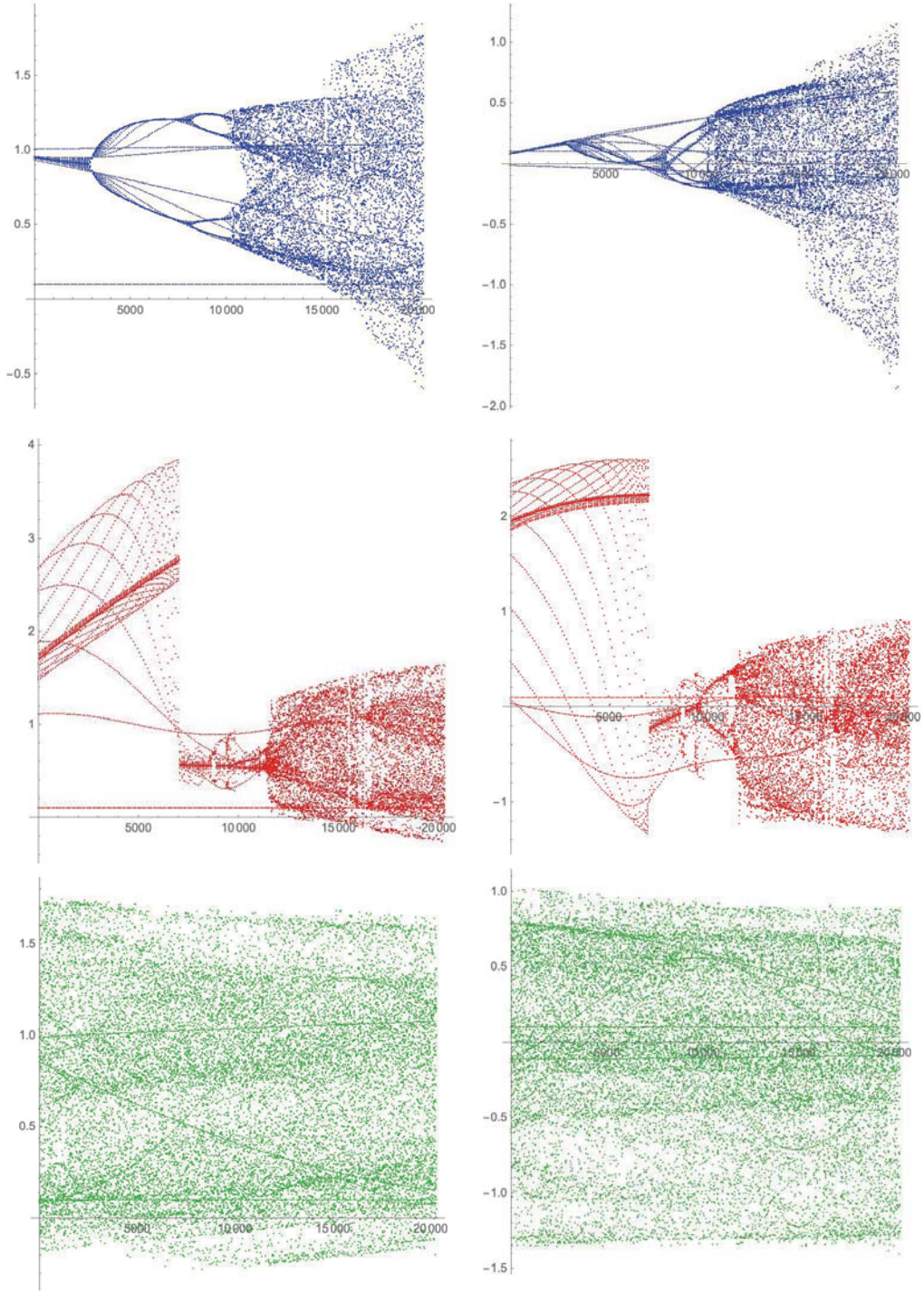


Figura 69: diagrama de bifurcaciones de Ikeda

2.3.4 - Mapas de dimensión 3

Estos mapas poseen tres órbitas de salida. Son sistemas dinámicos continuos (a diferencia de los vistos hasta aquí que eran discretos), por lo tanto, sus fórmulas están expresadas a partir de ecuaciones diferenciales. En el caso de la presente tesis el método escogido para resolver estas ecuaciones es el llamado "método de Euler" (Feldman, 2012), en donde de una ecuación de tipo $\frac{dx}{dt} = a\ x_n^2$ se puede obtener su órbita al calcular $x_{n+1} = x_n + \left(a\, x_n^2\right) \Delta t$. Del valor de Δt dependerá el grado de precisión obtenido. Calcular el resultado por otro método diferente al de Euler puede arrojar un resultado exacto, aunque esto requiere técnicas de cálculo que exceden a las necesidades de este trabajo.

Atractor de Lorenz

El atractor de Lorenz es un sistema de ecuaciones diferenciales ordinarias estudiado primero por Edward Lorenz (1917 - 2008). Es notable por tener soluciones caóticas para ciertos valores de parámetros y condiciones iniciales. Este sistema se conoce a nivel popular (e incluso en ciertos films) como el "efecto mariposa". Esto se deriva del hecho que, en el mundo real, en ausencia de un conocimiento cabal de las condiciones iniciales de un evento futuro, cualquier alteración en dichas condiciones iniciales puede desencadenar una serie de acontecimientos impredecibles. Es por esto que se ha hecho famosa la frase: "el aleteo de las alas de una mariposa puede provocar un Tsunami al otro lado del mundo"[80]. Esto subraya que los sistemas físicos pueden ser completamente

80 https://es.wikipedia.org/wiki/Efecto_mariposa

deterministas y aún ser inherentemente impredecibles. La forma del atractor de Lorenz en sí, cuando se representa gráficamente, también puede verse como una mariposa:

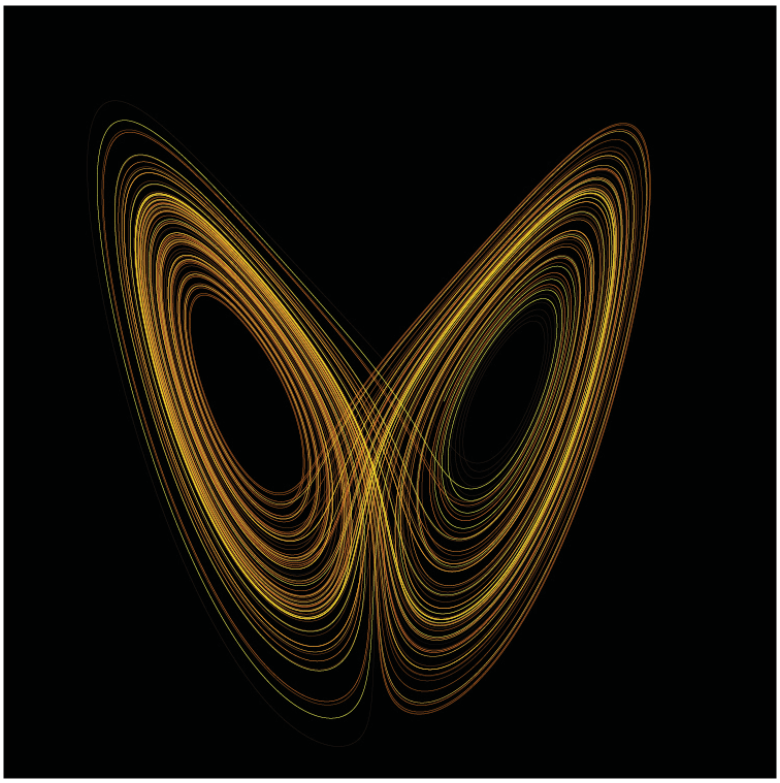


Figura 70: gráfico del atractor de Lorentz[81]

La ecuación para el cálculo de las órbitas es:

$$\frac{dx}{dt} = \sigma \ \left(y - x\right)$$

$$\frac{dy}{dt} = x\left(\rho - z\right) - y$$

$$\frac{dz}{dt} = x \ y - \beta \ z$$

Ecuación 19

81 En el que apreciamos su semejanza con la forma de una mariposa. Imagen descargada desde Wikipedia. User:Wikimol

Para implementar el método de Euler entonces, se procede de la siguiente forma:

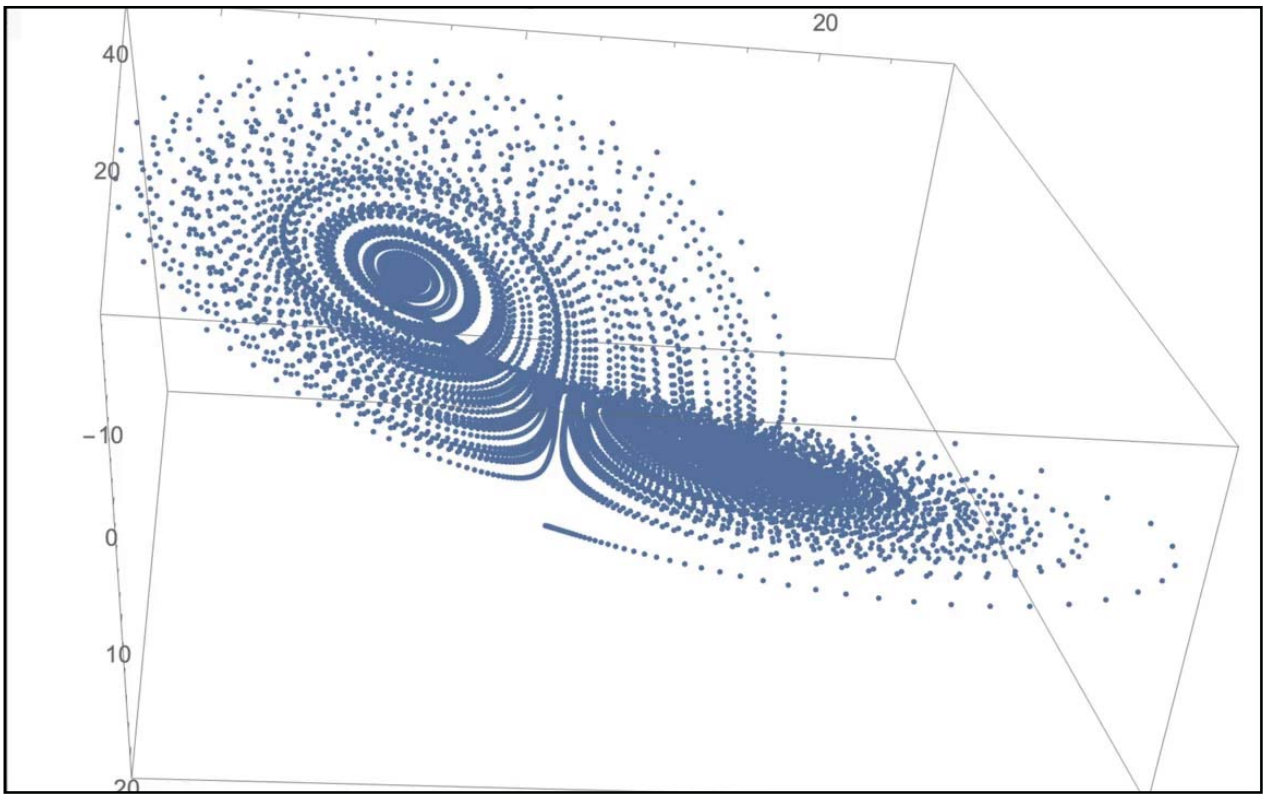
$$x_{n+1} = x_n + \left(\sigma\left(y_n - x_n\right)\right)\Delta t$$

$$y_{n+1} = y_n + \left(x_n\left(\rho - z_n\right) - y_n\right)\Delta t$$

$$z_{n+1} = z_n + \left(x_n y_n - \beta z_n\right)\Delta t$$

Ecuación 20

Esta ecuación, implementada en Mathematica, nos arroja el siguiente gráfico tridimensional:



Video 6: gráfico tridimensional de un atractor de Lorenz

En el caso de su diagrama de bifurcaciones se tendrán ahora tres órbitas y se podrían variar tres parámetros (σ, ρ, β) por lo tanto se obtendrían nueve gráficos. Para evitar colocar tantos de estos, se muestran solo algunos (figura 71) para tener una idea de cómo es el comportamiento de este mapa.

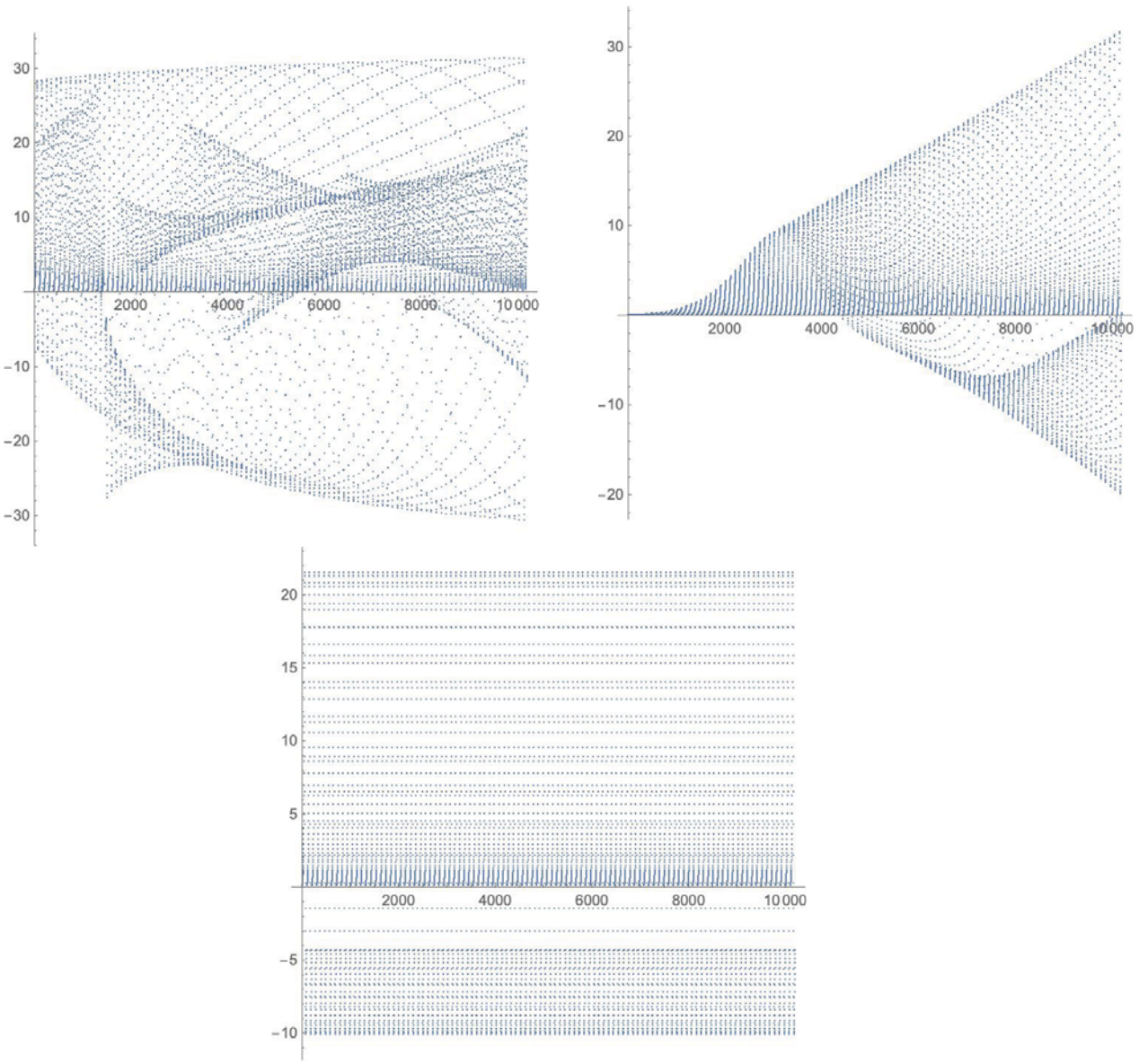


Figura 71: algunos diagramas de bifurcaciones de un atractor de Lorenz

Atractor de Rössler

Este atractor, al igual que el de Lorenz, está también conformado por un sistema de ecuaciones diferenciales de tres órbitas. Debe su nombre a Otto Rössler que lo propuso en un artículo publicado en *Physics Letters Journal* (Rössler, 1976). La ecuación es:

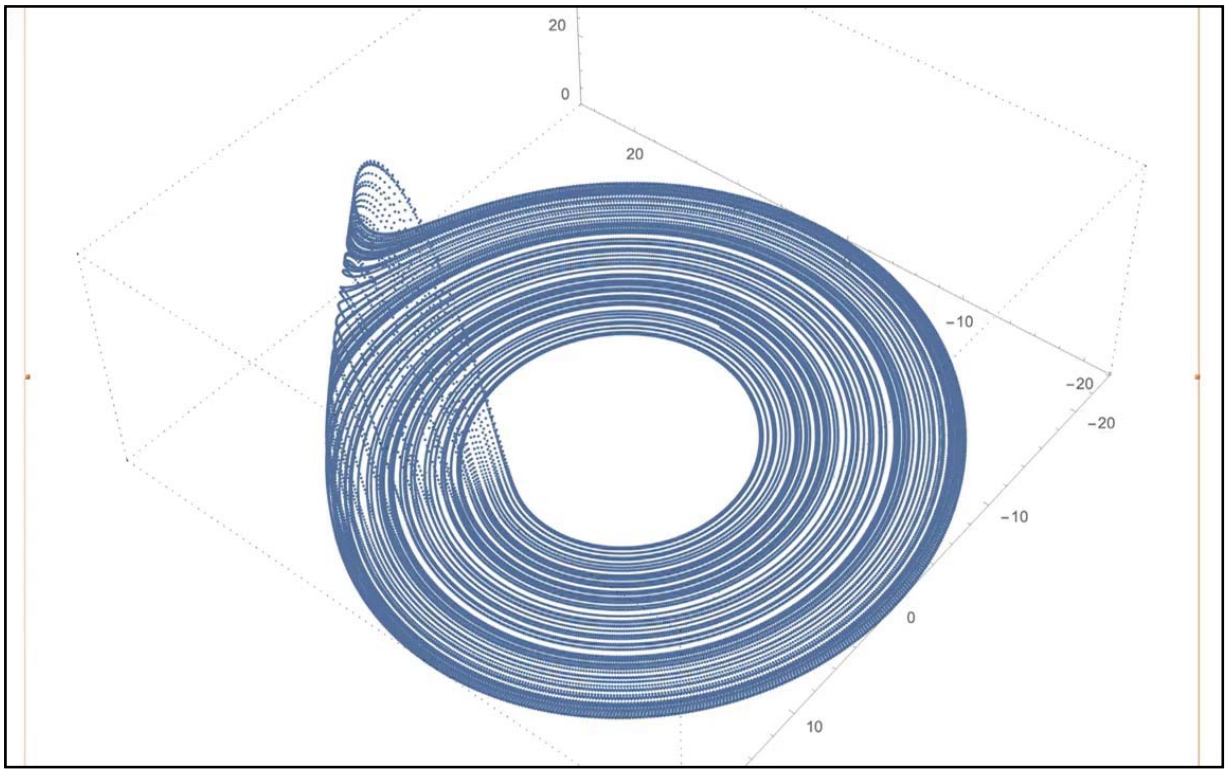
$$\frac{dx}{dt} = -\left(y + z\right)$$

$$\frac{dy}{dt} = x + ay$$

$$\frac{dz}{dt} = b + z\,(x - c)$$

Ecuación 21

A continuación se observa su gráfico en tres dimensiones:



Video 7: gráfico tridimensional de un atractor de Rössler

Al implementarlo en *Max* se hace necesario bajar la frecuencia de sampleo de la órbita ya que la distancia entre los sucesivos valores es demasiado pequeña. Se utilizó aquí un método para tomar uno de cada cien valores generados y el resultado parece más propicio para utilizarlo en el mapeo de parámetros. A continuación se observan algunos de sus diagramas de bifurcaciones:

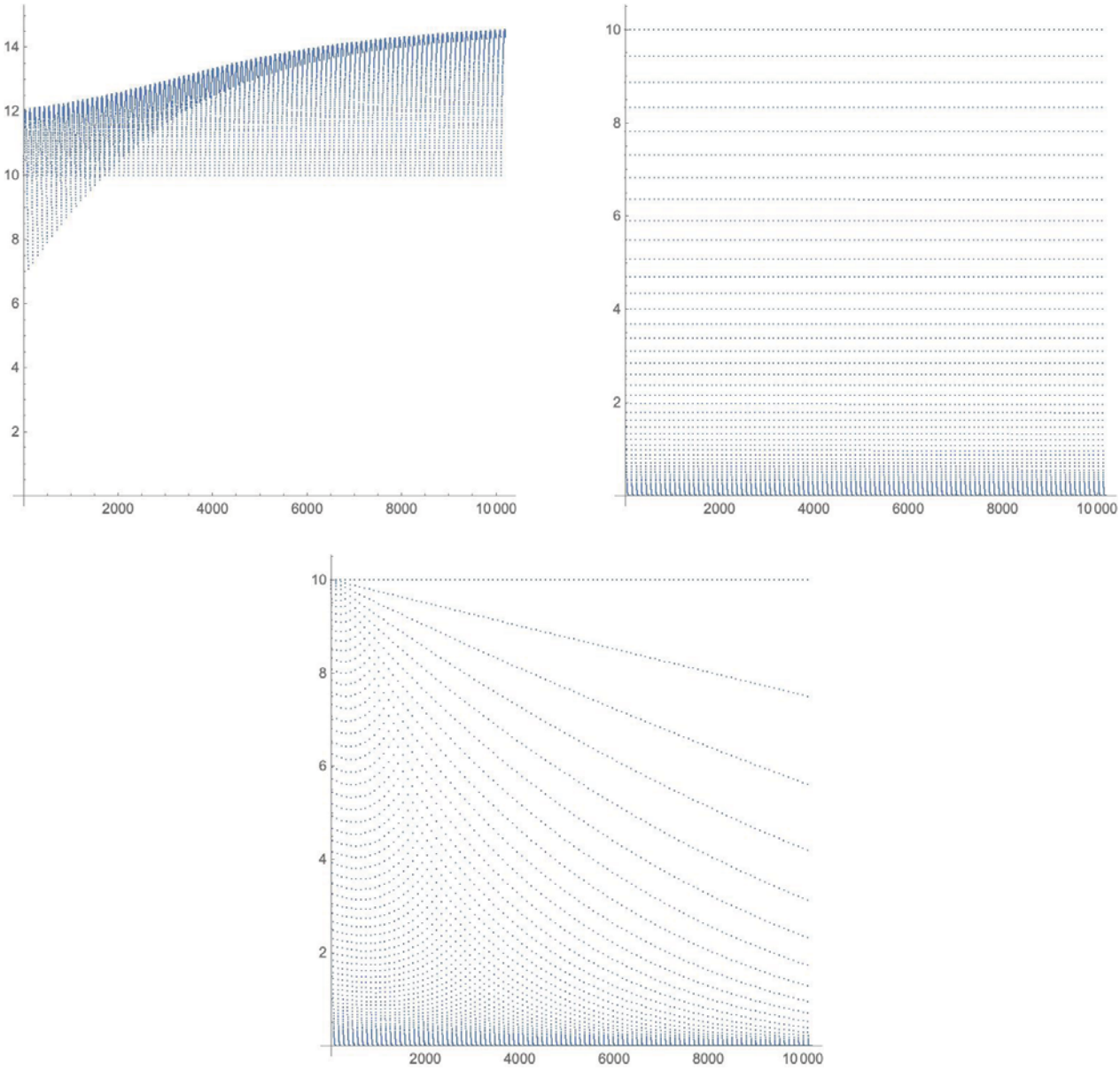


Figura 72: algunos diagramas de bifurcaciones de un atractor de Rössler

2.4 - Aleatorios

Como se mencionó anteriormente, los algoritmos para implementar estos generadores no utilizan la salida de la ecuación como entrada para el cálculo del siguiente valor[82].

Aquí se exponen diferentes tipos de procesos aleatorios de acuerdo a las probabilidades que posea este de generar ciertos valores o rangos de valores sobre otros. Por ejemplo, tirar un dado de seis caras (que no esté adulterado de ninguna forma) es un ejemplo de un generador aleatorio uniforme en el cual ningún valor tiene más probabilidades de ocurrir que otro. Sin embargo, existen otros modelos donde son más factibles las probabilidades de que un rango de valores aparezca con mayor frecuencia que otros. Aquí se exponen algunos de ellos, tomados de los propuestos por Charles Dodge (Dodge & Jerse, 1985) y Iannis Xenakis (Xenakis, 1971).

2.4.1 - Random walk

Este tipo de generación arroja dos valores posibles: 1 y -1. Partiendo de un número dado se acumulan uno de estos dos valores de manera aleatoria. Entonces si $x_0 = 0$ y se genera una secuencia como la siguiente: 1, 1, -1, 1, -1, -1, -1, 1, 1… se obtiene: 1, 2, 1, 2, 1, 0, -1, 1, 2. Es importante agregar a este proceso un limite superior e inferior desde los cuales el valor siempre “rebota” en sentido opuesto para que el sistema no se desborde. El típico gráfico de una función de random walk aleatoria es el siguiente:

82 Es por eso que se incluyó el generador linear congruencial entre los anteriores.

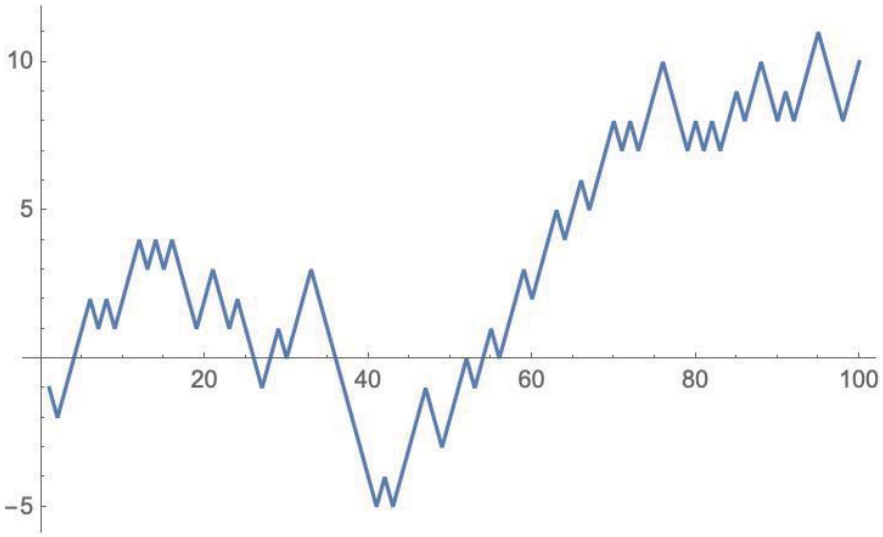


Figura 73: gráfico de un Random Walk

2.4.2 - Lineal

Este algoritmo genera valores aleatorios donde la posibilidad de ocurrencia disminuye a medida que el valor aumenta. Obtener esta distribución es bastante sencillo, se generan dos secuencias aleatorias diferentes y se escoge siempre el menor valor. De esta manera se obtiene el siguiente histograma:

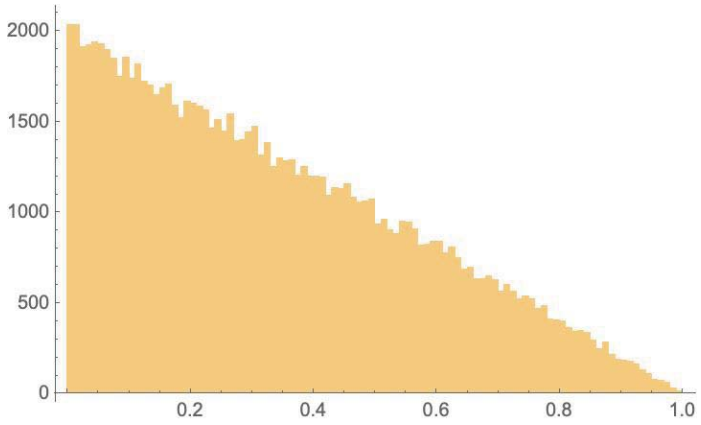


Figura 74: histograma de un generador aleatorio lineal

Es claro, por la figura 74, que los valores obtenidos son más probables a medida que el valor es menor. Si se modifica el algoritmo para escoger siempre el valor mayor, la pendiente será inversa.

2.4.3 - Triangular

Esta distribución pondera los valores intermedios y disminuye a medida que se va hacia los extremos. De allí su nombre. Para generarlo se utiliza el siguiente algoritmo:

$$\alpha = random$$
$$\beta = random$$
$$x_n = \frac{1}{2}(\alpha_n + \beta_n)$$

Ecuación 22

2.4.4 - Exponencial

Es similar a la distribución lineal donde los valores aleatorios cercanos al 0 son más probables que los cercanos al 1. Aquí, sin embargo, como su nombre lo indica, la curva de caída es exponencial:

$$\alpha = random$$
$$x_n = \frac{-Log(\alpha)}{\lambda}$$

Ecuación 23

2.4.5 - Bilateral

Esta distribución es a la triangular lo que la exponencial es a la lineal. Es decir, es una distribución triangular, pero con sus pendientes exponenciales.

$$\alpha = random\ (0-2)$$

$$f(x) = \begin{cases} \dfrac{-Log(2-\alpha)}{\lambda}, & 0 < x < 1 \\ \dfrac{Log(\alpha)}{\lambda}, & 1 \leq x < 2 \end{cases}$$

Ecuación 24

2.4.6 - Gauss

Esta es una distribución muy conocida a la que también se la llama distribución normal. Una distribución gaussiana se puede aproximar mediante la suma de números aleatorios distribuidos uniformemente (Dodge & Jerse, 1985). Esta distribución es la que aparece con mayor frecuencia al realizar un muestreo estadístico. Por ejemplo, si se toma el coeficiente intelectual de una determinada población se obtiene una distribución de este tipo. Para generar esta distribución se utiliza la siguiente fórmula:

$$\sigma * (\sqrt{-2\ Log(random)} * sin(2\ \pi\ random) + \mu$$

Ecuación 25

2.5 - Genéticos o evolutivos

Los procesos genéticos conforman una clase especial dentro de los algoritmos evolutivos, es decir, que son estrategias moldeadas a partir de sistemas naturales (Nierhaus, 2009). Estos sistemas naturales están inspirados en los trabajos de Charles Darwin (1809 - 1882) acerca de la evolución de las especies (Darwin, 1859). Dice Mitchell Melanie en su libro *An Introduction to Genetic Algorithms* (Mitchell, 1996):

> En los años cincuenta y sesenta, varios informáticos estudiaron independientemente sistemas evolutivos con la idea que la evolución podría usarse como una herramienta de optimización para problemas de ingeniería. La idea en todos estos sistemas era que se debía desarrollar una población de soluciones candidatas para un problema dado, utilizando operadores inspirados por variación genética y selección natural.

Es decir que se desarrollaron como un método para encontrar y resolver problemas utilizando procesos informáticos. En este tipo de algoritmos, entonces, los conceptos de *cromosoma*, s*upervivencia del más apto*, *herencia genética*, *cruzamiento*, *combinación* y *mutación* son importantes. Un cromosoma se representa aquí como una cadena de valores binarios. El procedimiento es el siguiente: se crea una población de *n* cromosomas (01100, 11011, 01011… etc), a partir de esta población, entonces, se calcula la aptitud de cada elemento y los más aptos son sometidos a cruzamientos y mutaciones. Este proceso se puede esquematizar de la manera que se observa en la figura 75.

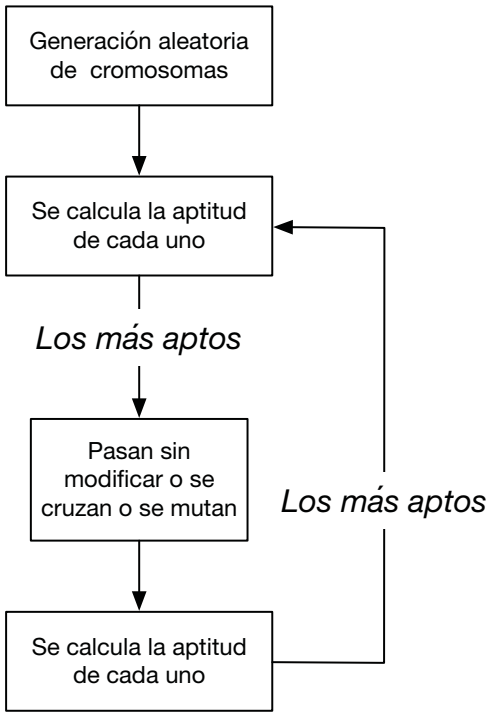


Figura 75: proceso para implementar un algoritmo genético

A continuación se ilustra este proceso con el ejemplo que propone David Goldberg en su libro *Genetic algorithms in search, optimization, and machine learning* (Goldberg, 1989) y que también utiliza Nierhaus (Nierhaus, 2009). Se comienza con una función que puede ser alimentada con valores de 5 bits, es decir, 32 valores posibles (0 ~ 31). Esa función es, para este ejemplo $f(x) = x^2$. Se toma un grupo de cromosomas generados al azar. Por ejemplo:

(01101) (11000) (01000) (10011)

A continuación se buscan cuáles de estos cromosomas son los más aptos. En este caso esa aptitud es completamente arbitraria ya que no se está lidiando aquí con las aptitudes de una determinada especie para sobrevivir en un entorno dado. En este caso, se ha medido la aptitud de cada cromosoma a partir del valor decimal que genera al atravesar la función $f(x) = x^2$

Nº	Cadena	Aptitud (valor decimal)	% del Total
1	1101	169 (ya que 01101 = 13, y $13^2 = 169$	14.4
2	11000	576	49.2
3	1000	64	5.5
4	10011	361	30.9
Total		1170	100

Tabla 1: ranking, a partir de $f(x) = x^2$, de los cromosomas generados

Luego cada cromosoma es distribuido en lo que Goldberg llama una “ruleta” (figura 76) donde cada uno de ellos posee una probabilidad de ser escogido de acuerdo a su aptitud. Así, el cromosoma 2 tiene mayor probabilidad de ser escogido que los demás, mientras que el cromosoma 3 es el menos probable.

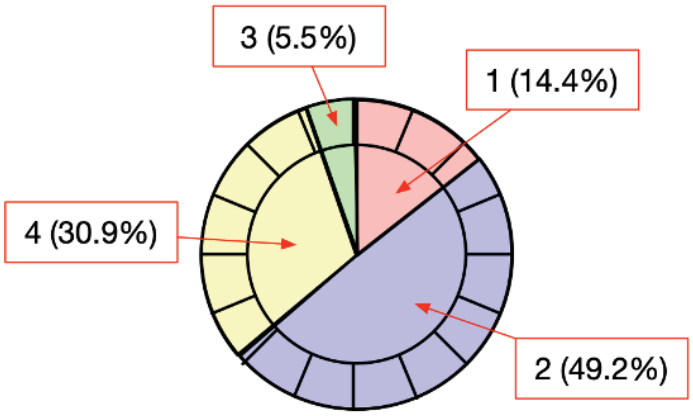


Figura 76: ruleta de Goldberg

El proceso comienza entonces seleccionando el cromosoma de manera aleatoria pero tomando en cuenta el “peso” de cada valor, es decir la probabilidad de ser seleccionado (en este caso el porcentaje de aptitud con respecto al total). Si, utilizando este procedimiento, se seleccionan cuatro cromosomas: una vez el número 1, dos veces el número 2 y una vez el número 4 (el 3 no sale seleccionado por su bajo porcentaje de aptitud), entonces quedará:

01101

11000

11000

10011

Ahora los cromosomas seleccionados son “apareados” escogiéndolos de manera aleatoria, por ejemplo:

01101 → 11000

El proceso para el apareamiento es el siguiente: se escoge un valor de índice aleatorio entre 1 y el largo de la cadena menos 1. Es decir que, en este caso, como la cadena de números binarios tiene 5 valores, se escogerá un valor entre 1 y 4. En el caso de que dicho valor sea 2, entonces el cruce se realizará intercambiado los valores de un cromosoma por el otro de la siguiente manera:

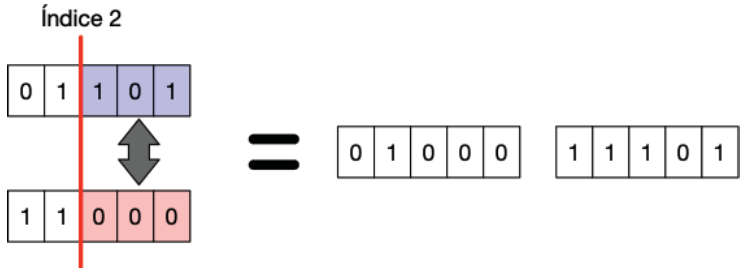


Figura 77: proceso de apareamiento

Si este proceso se repitiera dos veces:

Random 1	Random 2	Resultado 1	Resultado 2
11000	01101	11101	01000
10011	11000	10000	11011

Tabla 2: proceso de cruce

Al calcular sus aptitudes:

Nº	Cadena	Aptitud (valor decimal)	% del Total
1	11101	841	44.5
2	01000	64	3.4
3	10000	256	13.5
4	11011	729	38.6
Total		1890	100

Tabla 3: tabla de aptitudes

David Goldberg explica en el libro mencionado mas arriba que este proceso, aunque involucra el uso de selecciones aleatorias, es muy eficiente a la hora de encontrar soluciones de manera automática:

> Las mecánicas de la reproducción y el crossover son sorprendentemente simples, implican la generación de números aleatorios, copias de cadenas y algunos intercambios parciales de las mismas. No obstante, el énfasis combinado de la reproducción y el intercambio estructurado, aunque aleatorio, de información de cruce proporcionan a los algoritmos genéticos gran parte de su poder. Al principio esto parece sorprendente. ¿Cómo pueden dos operadores tan simples (y triviales computacionalmente) resultar en algo útil, y mucho menos en un mecanismo de búsqueda rápido y robusto? Además, ¿no parece un poco extraño que la aleatoriedad juegue un papel tan fundamental en un proceso de búsqueda dirigida? (Goldberg, 1989)

A continuación de este procedimiento toma lugar el proceso de mutación donde algún valor de la cadena es modificado:

11[1]01 → 11[0]01

Entonces, el proceso descripto en la figura 75 puede completarse así:

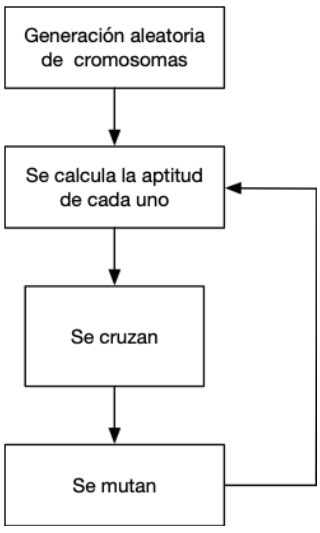


Figura 78: extensión del proceso señalado en la figura 75

2.5.1 - Técnicas de selección

Como se mencionó, en la fase de selección se determina qué individuos son escogidos para aparearse. Esto se realiza a partir del concepto de aptitud, siendo esa aptitud el resultado de sopesar cada cromosoma al atravesar una función dada. Sin embargo, existen diferentes técnicas para realizar esta selección basadas en la aptitud y la probabilidad. Algunas de estas técnicas se detallan a continuación.

Selección tipo ruleta

Esta es la técnica que se mostró en la figura 76. Aquí la probabilidad de ser escogido es proporcional al área asignada a cada cromosoma dentro de la rueda, y esa área estará dada por el porcentaje de aptitud obtenido por cada miembro al atravesar una función específica. En el caso de la figura 76 la función es $f(x) = x^2$. Así entonces la probabilidad de de ser escogido es:

$$p_i = \frac{f_i}{\sum_1^w f_i}$$

Ecuación 26

Siendo p_i el cromosoma, f_i el valor de aptitud del cromosoma i^{th} y *w* el tamaño de la población.

Este sistema tiene la desventaja de que, si la diferencia entre los cromosomas más fuertes y los más débiles es muy grande, los cromosomas débiles tienen muy poca o ninguna chance de ser escogidos. Es decir que, al seleccionar un cromosoma de manera aleatoria,

pero tomando en consideración la probabilidad de selección de cada uno o el “peso”, se podría obtener una selección como la de la figura 79.

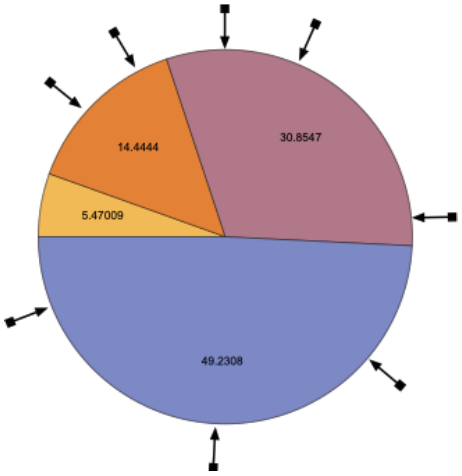


Figura 79: selección de tipo ruleta

En ese tipo de selección puede darse el caso que después de ocho chances ninguna caiga sobre la porción más pequeña. Una manera de evitar esto es utilizar lo que Baker llama *muestra estocástica universal* (Baker, 1987) en la cual se selecciona al azar un punto de la rueda y luego los siguientes puntos se toman de manera equidistante con una proporción igual a 1/Nº de muestras (figura 80).

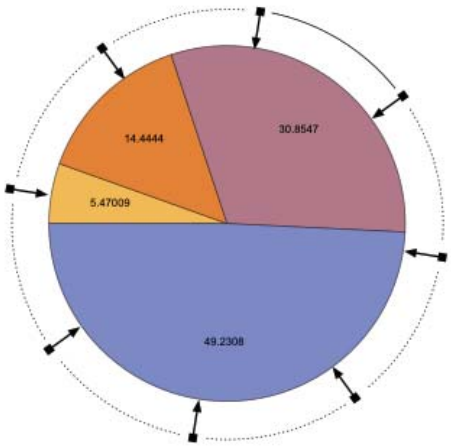


Figura 80: muestra estocástica universal

Selección por ranking lineal

Aquí los cromosomas son listados en un ranking en el cual el más débil tendrá un valor de 1 y el más fuerte un valor de N. Usando la población anterior esto quedaría:

Nº	Cadena	Aptitud	% del Total	Ranking
1	1101	169	14.4	2
2	11000	576	49.2	4
3	1000	64	5.5	1
4	10011	361	30.9	3

Tabla 4: selección por ranking lineal

La fórmula para calcular la probabilidad de p_i de ser escogido es:

$$\frac{1}{N}(\varphi^- + (\varphi^+ - \varphi^-)\frac{i-1}{N-1}$$

Ecuación 27

Donde *N* es el total de la población, φ^+ es la probabilidad del más apto de ser seleccionado comparado con la probabilidad promedio de selección de todos los individuos (este valor siempre está entre 1 y 2), $\varphi^- = 2 - \varphi^+$ y, por último, *i* es el puesto en el ranking de cada individuo. Por ejemplo, utilizando los cromosomas de la Tabla 4 (*N* = 4):

$$\varphi^+ = \frac{N \,/\, \sum i}{1 \,/\, N} = \frac{4/10}{1/4} = 1.6$$

$$\varphi^- = 2 - \varphi^+ = 2 - 1.6 = 0.4$$

Entonces

$$\frac{1}{4}(0.4 + (1.6 - 0.4)\frac{i-1}{4-1}$$

Así los valores obtenidos son:

$$i_1 = 0.1 \quad i_2 = 0.2 \quad i_3 = 0.3 \quad i_4 = 0.4$$

Al comparar la distribución utilizando la técnica de ruleta versus la de ranking lineal se observa que la segunda tiene una distribución más proporcional, haciendo que los cromosomas más débiles tengan mas oportunidad de ser seleccionados.

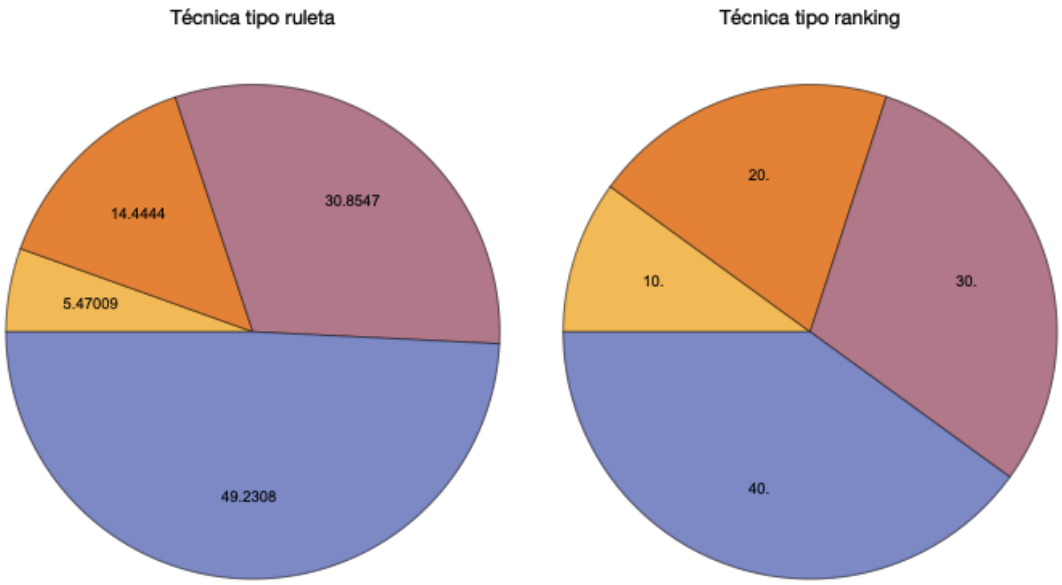


Figura 81: resultados obtenidos a partir de la técnica de ruleta y la de ranking lineal

Selección por ranking exponencial

Como su nombre lo indica, esta técnica es una variación de la selección por ranking lineal en la cual es posible distribuir las probabilidades de cada cromosoma de manera

exponencial. La base del exponente es *c* , donde *0 < c < 1* (Shukla, Pandey, & Mehrotra, 2015). La probabilidad de p_i, entonces esta dada por:

$$\frac{c^{N-i}}{\sum_{j=1}^{N} c^{N-j}}$$

Ecuación 28

Donde *N* es el total de la población y *c* es la base del exponente (con la que se controla la curva deseada).

Selección tipo torneo

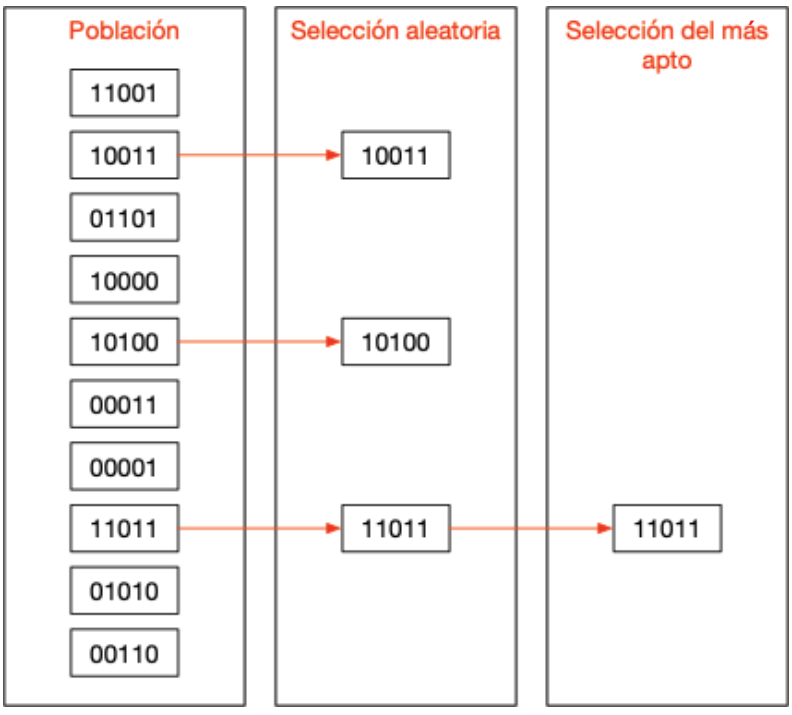


Figura 82: selección tipo torneo

Este tipo de selección es una combinación de la aleatoria y por aptitud (figura 82). De una población determinada se escogen una serie de individuos al azar que serán los

competidores. Luego se selecciona de entre estos al más apto. Este procedimiento se repite cuantas veces sea necesario hasta escoger un cromosoma.

2.5.2 - Técnicas de cruzamiento o recombinación

Como se vio en la figura 75, una vez seleccionada la población a partir de alguna de las técnicas de selección vistas anteriormente, se procede a recombinar diferentes pares de cromosomas. Uno de los métodos más comunes para realizar esta tarea es el que se muestra en la figura 77 llamado *cruce de 1 punto* (Umbarkar & Sheth, 2015), pero existen otras técnicas para llevar a cabo la misma tarea. A continuación, algunas de ellas.

Cruzamiento de K puntos

Se seleccionan varios puntos al azar donde realizar la recombinación. Ver el siguiente ejemplo:

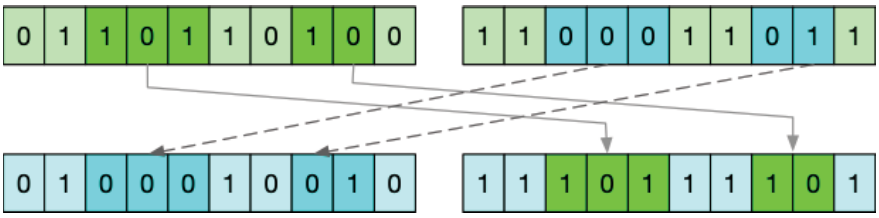


Figura 83: cruzamiento de K puntos

Este procedimiento se aplica, con frecuencia, con no más de dos grupos por cromosoma, aunque esto dependerá cuán largo sea la cadena de valores binarios.

Recombinación barajando

Se toman dos cromosomas y se barajan sus elementos en el mismo orden como se ve, por ejemplo, en la figura 84.

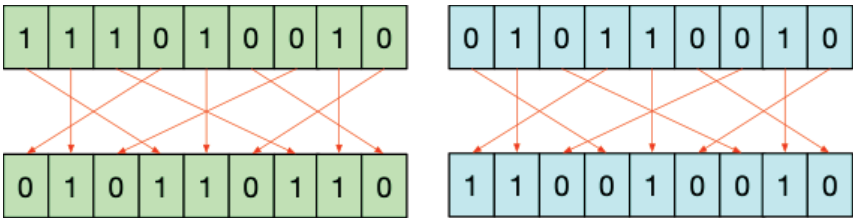


Figura 84: recombinación barajando parte 1

Luego se realiza un cruce de 1 punto:

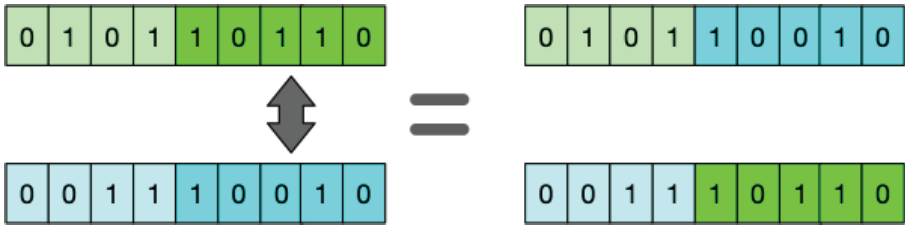


Figura 85: recombinación barajando parte 2

Y por último de “des-barajan” de la misma manera que antes se barajaron

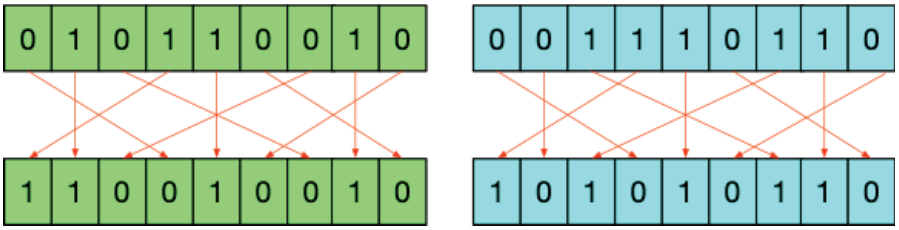


Figura 86: recombinación barajando parte 3

Recombinación uniforme

Aquí se comparan los bits de cada cromosoma y, los que son distintos, se intercambian con una probabilidad aleatoria dada. Si esa probabilidad es de exactamente del 50% se lo denomina “recombinación uniforme media”.

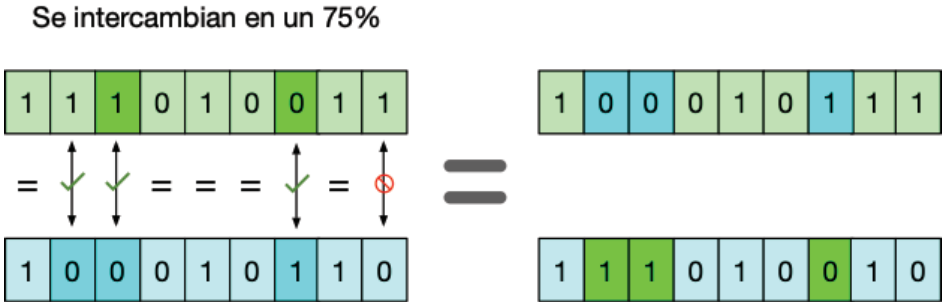


Figura 87: recombinación uniforme

Además de estas técnicas es posible utilizar, entre otras, el cruzamiento enmascarado, el cruzamiento promedio, el cruzamiento heurístico (Umbarkar & Sheth, 2015)

2.5.3 - Técnicas de mutación

La mutación se puede definir como un pequeño ajuste aleatorio en el cromosoma, para obtener una nueva solución. Se utiliza para mantener e introducir diversidad en la población genética[83]. La mutación de un cromosoma implica la creación de una variación del mismo alterando, de manera aleatoria, una parte (seleccionada también de manera aleatoria) del parental (Poli, Langdon, & McPhee, 2008). Algunas de las técnicas utilizadas para tal fin se detallan a continuación.

Mutación por bit contrario

Aquí se selecciona al azar uno o mas bits del cromosoma y se los reemplaza por el valor

83 https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_mutation.htm

contrario. Ver figura 88.

Figura 88: mutación por bit contrario

Mutación por intercambio

Como lo indica su nombre, se intercambian dos bits de dos cromosomas escogidos aleatoriamente. En la codificación de los cromosomas en números binarios (como los vistos hasta aquí) no hay diferencia entre esta técnica y la anterior, pero si la codificación es diferente (como efectivamente sucede en el estudio del ADN) esta técnica cobra mayor sentido:

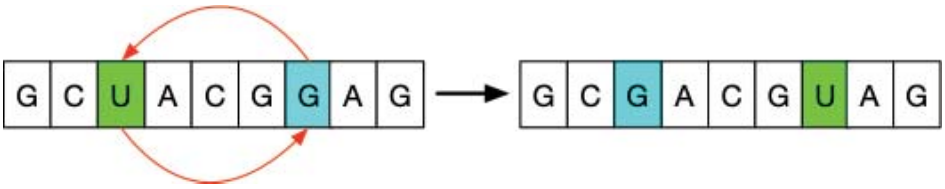


Figura 89: mutación por intercambio

Mutación barajando

Se selecciona una porción del cromosoma al azar y dentro de esta región se mezclan las posiciones de sus bits de manera aleatoria.

Figura 90: mutación barajando

Como sucede con la recombinación, existen varias técnicas de mutación diferentes. Hacer una lista completa de ellas excede el propósito de este trabajo.

2.6 - Autómatas celulares

Las primeras teorías sobre autómatas celulares en el sentido más estricto fueron propuestas en la década de 1950 por Konrad Zuse, Stanislav Marcin Ulam y John von Neumann (Nierhaus, 2009). Los autómatas celulares son sistemas dinámicos discretos arreglados en forma de grilla. Estas variables se actualizan a intervalos de tiempo discreto a partir de ciertas reglas locales determinísticas (Feldman, 2012). Dado que los autómatas celulares (CA por su nombre en inglés) producen grandes cantidades de patrones, no sorprende que los compositores hayan comenzado a sospechar que podrían ser una buena herramienta para generar material musical (Miranda & Biles, 2007). El estado de una celda en un momento de tiempo discreto t_0 está determinado por su propio estado, así como por los estados de las celdas vecinas en el momento t_{n-1}. En este sentido los CA se relacionan con los sistemas recursivos que se expusieron en la sección 2.3. Estos sistemas se pueden clasificar según su dimensión. A continuación, una breve descripción de cada uno de ellos.

2.6.1 - Autómatas celulares 1D

Un sistema CA donde cada celda puede tener dos estados y una cantidad de celdas *n*, se denomina CA binario (Feldman, 2012) y se representa de la siguiente manera:

Figura 91: representación mas utilizada de una CA de 1D

Es obvio que este sistema puede ser una manera de representar un cromosoma como los expuestos en el punto 2.5, pero las reglas de reproducción y combinación aquí son diferentes. Dichas reglas establecen que el valor posterior de una celda estará determinado por su valor actual y por los valores de las celdas vecinas, una a la izquierda y la otra a la derecha. Estas se representan de la siguiente manera:

Figura 92: representación de las reglas de evolución de un CA

En la figura 92 se observa un gráfico que representa una regla determinada, en donde cada grupo de tres casilleros tiene debajo el resultado que se obtendrá para cada combinación (con 3 células se obtienen 8 combinaciones). Así se estipula que tres cuadros negros generarán un cuadro banco, los dos primeros cuadros negros y el tercero blanco generan, también, un cuadro blanco, pero un cuadro negro y los dos siguientes blancos generan un cuadro negro. Aplicando esta regla a la siguiente cadena inicial, tomando en consideración que el vecino izquierdo de la primera celda es la última celda de la derecha y el vecino derecho de la última celda es la primera celda (como se hizo con los *L-Systems*):

Figura 93: ejemplo de CA utilizando las reglas de la figura 88

Así quedaría:

Figura 94: resultado obtenido, grupo figura 93 y reglas de la figura 92

Esto se ve más claro si se divide la cadena inicial en grupos de tres celdas:

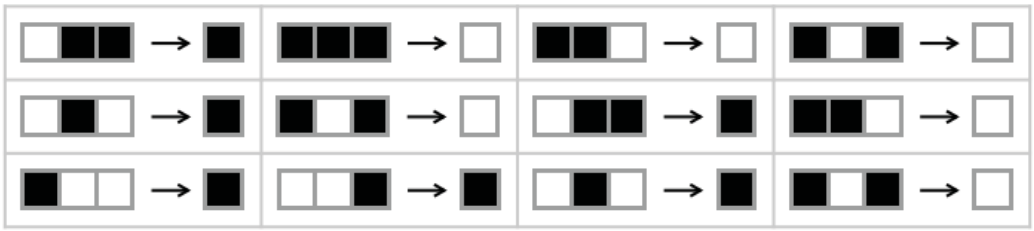


Figura 95: grupos de tres celdas

Si se itera este proceso 10 veces se obtendrá un diagrama espacio-temporal como el que se muestra en la figura 96. Es claro entonces que el resultado obtenido depende de la regla que se utilice y la configuración de partida. Por ejemplo, si se utiliza la misma regla de la figura 92 pero se comienza con una sola célula con valor 1, se obtendría lo que se observa en la figura 97.

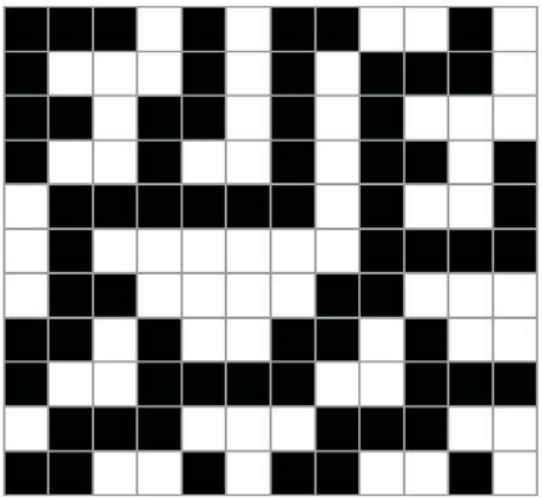


Figura 96: resultado después de 10 iteraciones

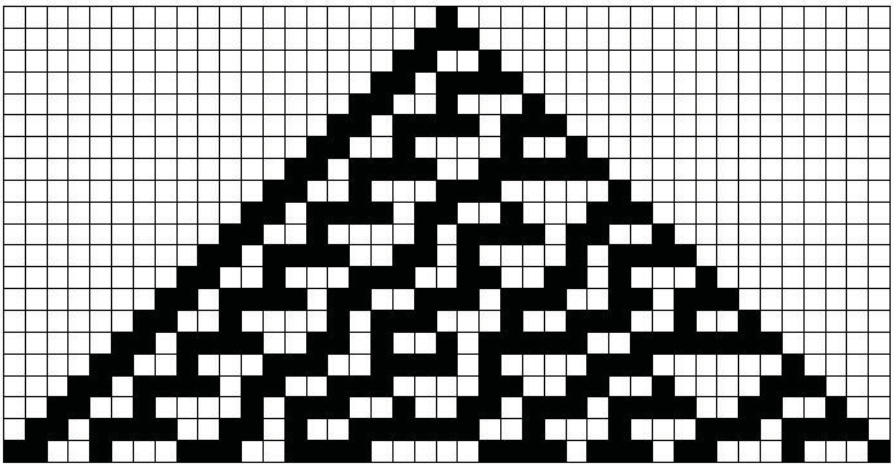


Figura 97: comenzando con una sola célula de valor 1

Las reglas para CA de tres miembros son 256 diferentes ya que hay 8 posibles configuraciones que pueden adoptar el valor 0 o 1. Así la regla 30 sería: 0001110, es decir, que cada regla recibe su valor de su numero binario resultante. El número binario 00011110 es igual a 30 en sistema decimal. Por ejemplo, ya que 120 en binario es igual a 01111000, la regla 120 tendrá la siguiente configuración:

Figura 98: regla 120

Es necesario aclarar que, al combinar una regla determinada con un punto de partida dado, puede suceder, después de *n* iteraciones, que se alcance un punto fijo (como lo visto en el caso del mapa logístico) a partir del cual las iteraciones siguientes ya no se modifican. Como ya se mencionó, el ejemplo de la figura 96 está formado por 10 iteraciones, pero si se sigue iterándolo (hasta 20, por ejemplo) se verá que a partir de la iteración 17 el CA ya no evoluciona. Es decir que aquí la condición inicial determina la

órbita del sistema dinámico. En este caso, la órbita es una secuencia de filas de variables discretas (Feldman, 2012).

Las combinaciones que se obtienen de la interacción de estas dos variables son diversas. Algunas llegan a un punto fijo después de pocas iteraciones (figura 99), otras lo hacen después de varias, otras generan un diagrama espacio-temporal periódico sin importar la condición inicial (como el de la figura 100) y otras generan un diagrama caótico (figura 101).

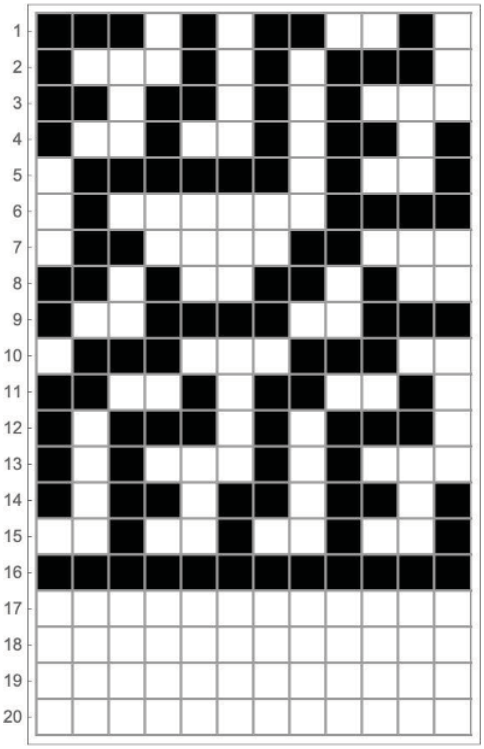


Figura 99: resultado después de 20 iteraciones del gráfico de la figura 96

En el caso de la figura 100 la regla utilizada es la 46 y en el caso de la figura 101 es la regla 150. También existen situaciones “intermedias” donde el resultado es caótico, pero se aprecia un componente de periodicidad. En este sentido la regla número 110 (figura 100) ha sido muy estudiada ya que, aparentemente, posee estas dos características

contradictorias (W. Li & Nordahl, 1992; Martinez et al., 2006; Neary & Woods, 2006).

Figura 100: diagrama espacio-temporal periódico

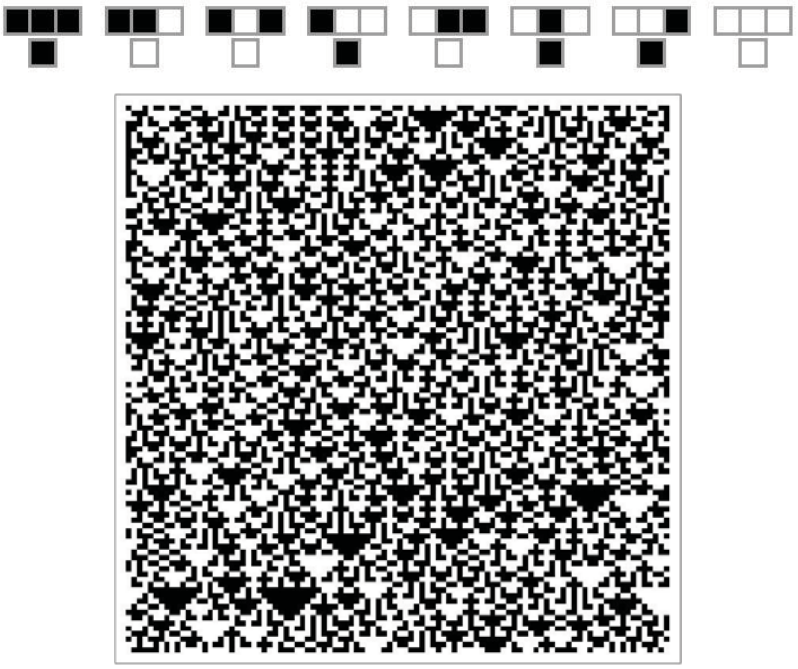


Figura 101: diagrama espacio-temporal caótico

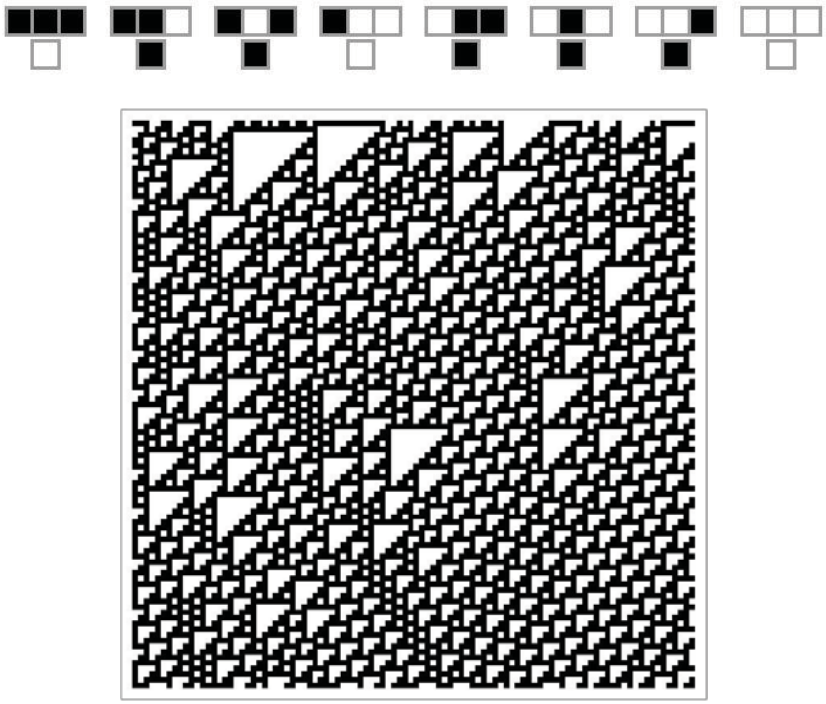


Figura 102: diagrama espacio-temporal regla 110

Radio del Autómata Celular (CA)

Los CA vistos hasta aquí son los más simples de todos y se denominan “autómata celular elemental”, ya que poseen una sola dimensión, dos posibles estados y un “vecindario” de 3 células al que se llama “radio” del CA y se denota con la letra *r* (Vázquez & Oliver, 2008). Sin embargo, estas características pueden modificarse. En el caso del radio se puede utilizar un vecindario tan grande como se desee, solo hay que tomar en consideración que cada nueva célula agrega una celda de cada lado del grupo, es decir que, si se agrega una celda a las vistas hasta aquí, se obtendría un grupo de 5 cuadros. Esto implica tener un número binario de 5 posiciones, lo que arrojaría 32 posibles estados obteniendo así 4,294,967,296 posibles reglas.

Estados del CA

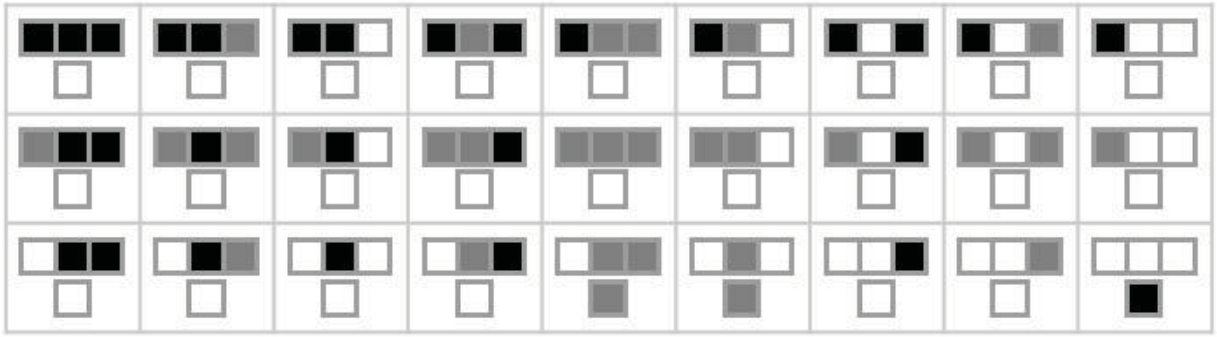


Figura 103: una de las 7,625,597,484,987 de reglas posibles de r =3 y K = 3

Al igual que con el radio, los posibles estados (denotado con la letra *K*) de una celda pueden ser más de dos (0 y 1). Aquí también hay que considerar las implicancias de esto en cuanto al número de reglas que se obtendrán. Si se posee un vecindario con r = 3 y K = 3, se tendrán entonces $3^3 = 27$ estados, lo que arroja $3^{27} = 7{,}625{,}597{,}484{,}987$ reglas posibles. Como se puede apreciar, pequeñas variaciones en *r* y en *K* provocan que estas posibilidades adquieran valores astronómicos.

2.6.2 - Autómatas celulares 2D

Aquí se toma en consideración no solo las celdas de los costados sino las que rodean a la célula central. En este sentido se puede tener un vecindario de 5 celdas (donde se consideran la central, la derecha, la izquierda, la superior y la inferior) o un vecindario de 9 células donde se consideran todos los cuadros que rodean al central (ver figura 104). A la primera se la denomina “vecindario de Von Neumann” y a la segunda “vecindario de Moore” (Wolfram, 1985). En el caso del vecindario de 5 celdas, si este es binario (es decir si cada celda solo puede adoptar el valor 0 o 1) se tendrán 2^5 diferentes estados que

generan $2^{2^5} = 4{,}294{,}967{,}296$ reglas diferentes, mientras que con el vecindario de 9 celdas se tendrían 2^9 estados que generarán 2^{2^9} reglas, lo que implica un numero enorme de posibilidades[84].

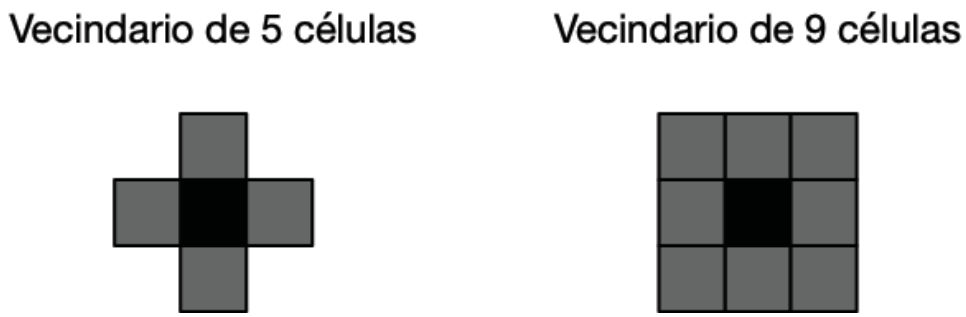


Figura 104: autómata celular 2D de 5 y 9 vecinos

El más conocido de los autómatas celulares binarios de 9 células es el llamado “Game of Life”, introducido por John Conway y popularizado por Martin Gardner (Gardner, 1971). Aquí las reglas son simples (McIntosh, 2010):

• Una célula viva sobrevive si tiene dos o tres vecinos vivos.

• Nace una nueva célula cada vez que hay tres vecinos vivos.

• Todas las demás células mueren o permanecen inactivas.

Al utilizar estas reglas en un CA de 2 dimensiones, algunas configuraciones iniciales evolucionan de maneras interesantes. Estas configuraciones reciben sus nombres a partir

84
13,407,807,929,942,597,099,574,024,998,205,846,127,479,365,820,592,393,377,723,561,443,721,764,030,073,546,976,801,874,298,166,903,427,690,031,858,186,486,050,853,753,882,811,946,569,946,433,649,006,084,096

de su forma o comportamiento. A continuación algunas de ellas[85]:

Naturaleza muerta

Se denominan así a las configuraciones iniciales que no evolucionan sino que se mantienen estáticas a lo largo de las diferentes iteraciones.

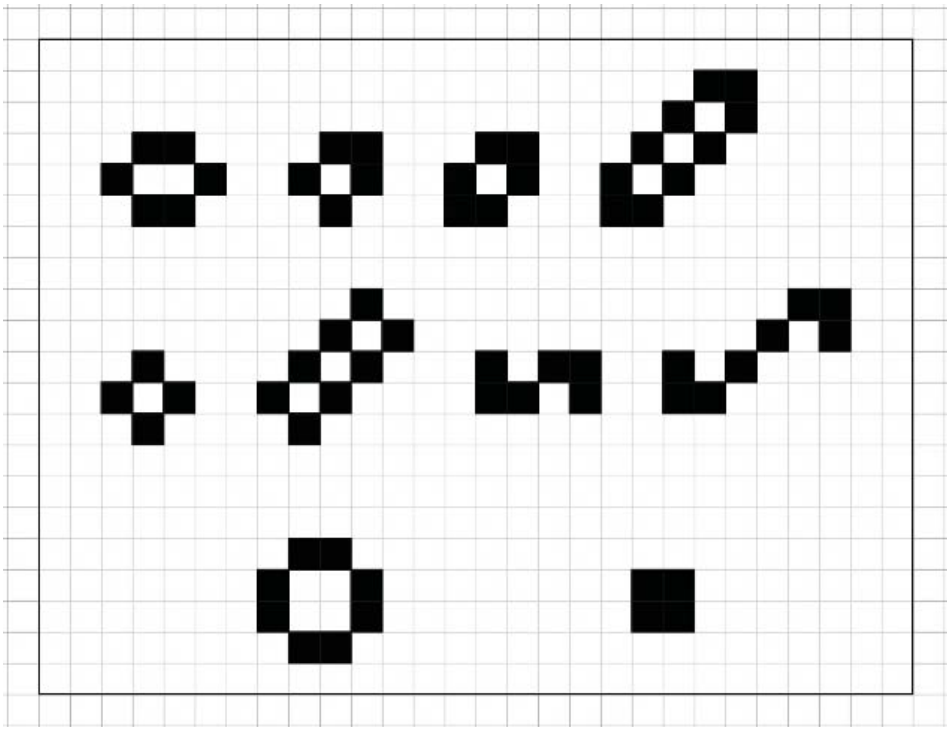


Figura 105: Game of Life, configuración. Inicial tipo “naturaleza muerta”

Tomando como ejemplo el caso de la configuración que se ve en la figura 105 inferior derecha (cuatro cuadros que forman un cuadro mayor) es sencillo entender el comportamiento de estas figuras. Si se considera a cada cuadro como el centro de un vecindario de Moore, entonces todas ellas tienen tres vecinos vivos:

85 Puede verse una lista extensa de formas en la web de Achim Flammenkamp. http://wwwhomes.uni-bielefeld.de/achim/freq_top_life.html

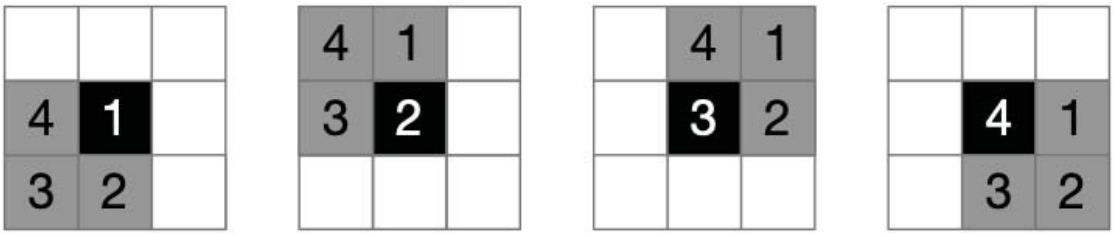
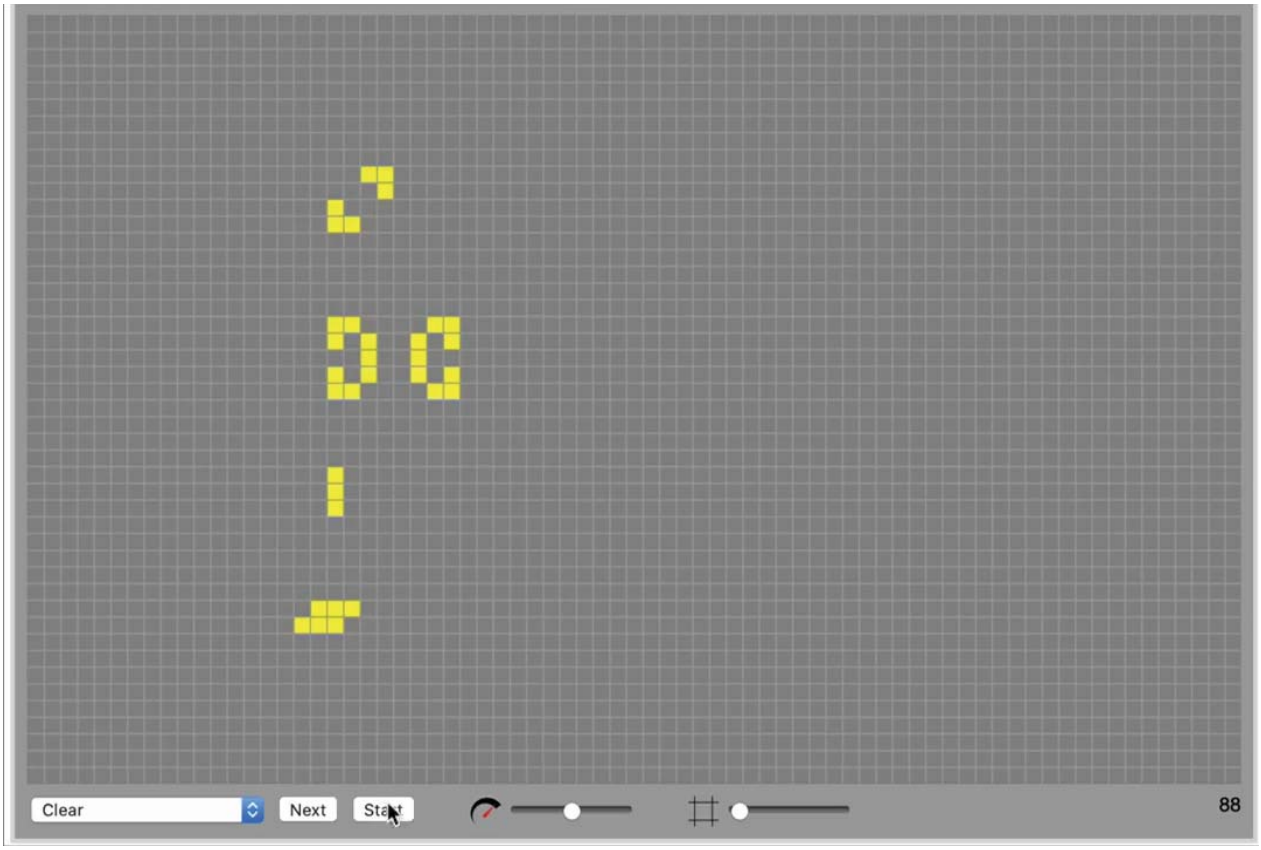


Figura 106: explicación de la evolución estática de una “naturaleza muerta”.

Osciladores

Estas son configuraciones iniciales que permanecen oscilando entre *n* estados diferentes. Las que oscilan entre dos estados, junto con los cuadrados de cuatro celdas, son las formas más comunes que quedan remanentes en una evolución temporal. En el video 8 se pueden observar algunas de estas formas iniciales y su oscilación en dos estados.



Video 8: osciladores. Simulación generada en la web https://bitstorm.org/gameoflife/

Otras configuraciones oscilan entre más estados que dos. Las más “vistas” según Achim Flammenkamp[86] son las de 3, 4, 5, 6, 8, 14, 15 y 30 ciclos. A continuación, se muestra un ejemplo de cada una:

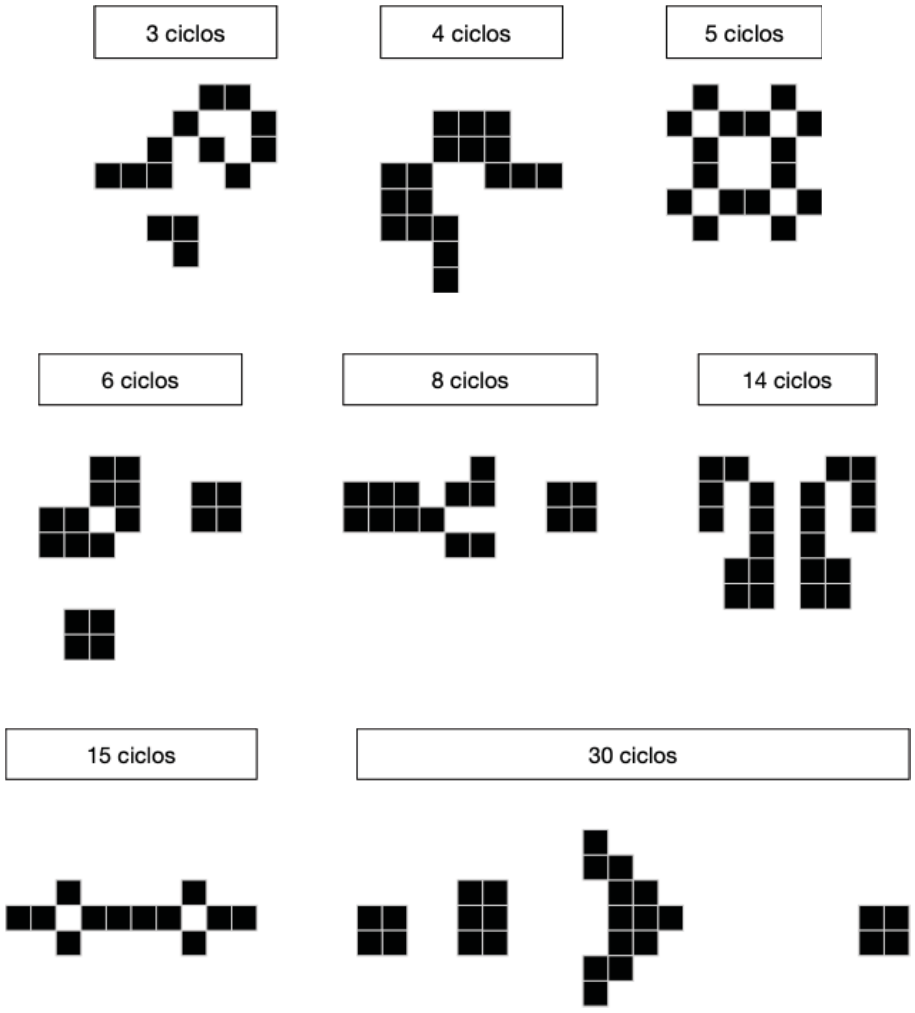


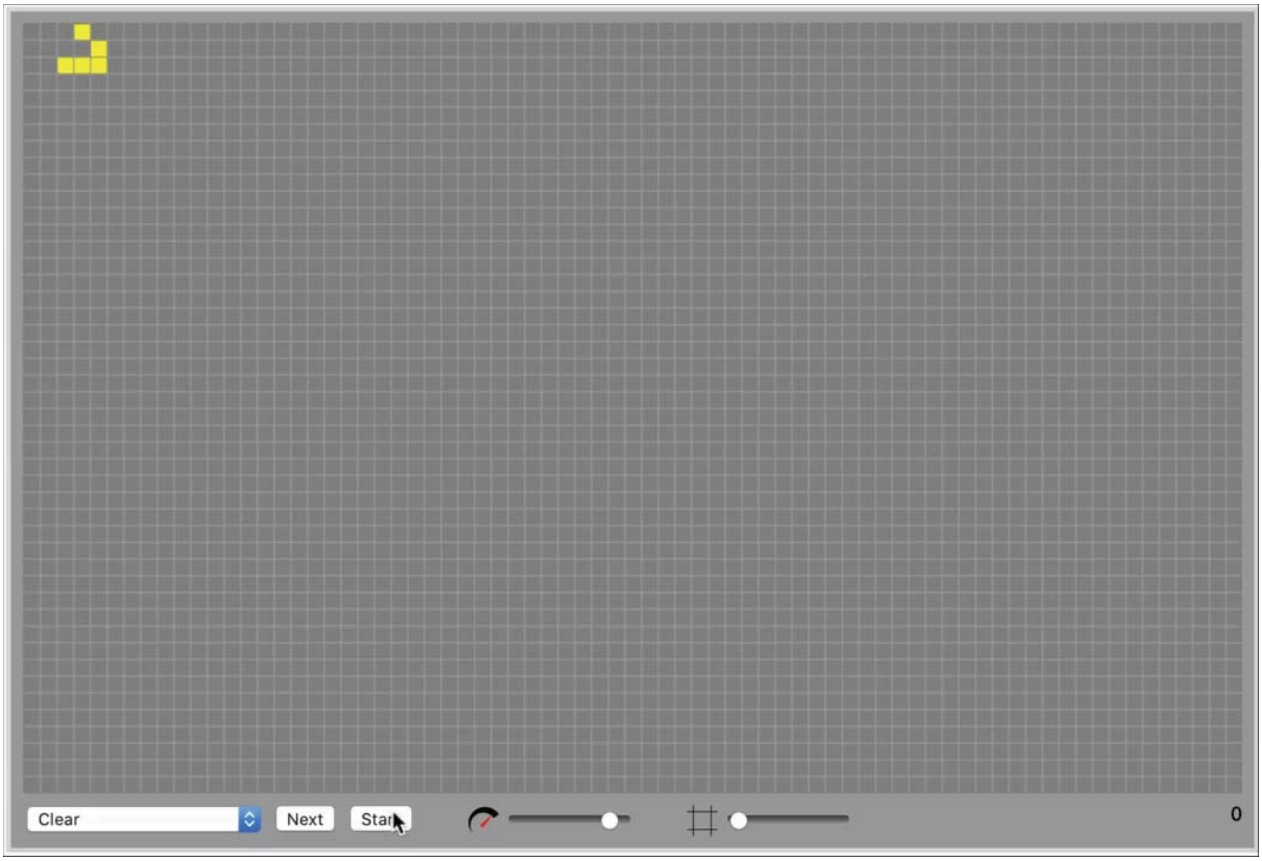
Figura 107: osciladores de 3, 4, 5, 6, 8, 14, 15 y 30 ciclos

Planeadores

Estas configuraciones se desplazan o planean en la cuadrícula, y de allí su nombre. A medida que recorren el espacio (que en teoría es infinito) van mutando su forma de

86 http://wwwhomes.uni-bielefeld.de/cgi-bin/cgiwrap/achim/index.cgi

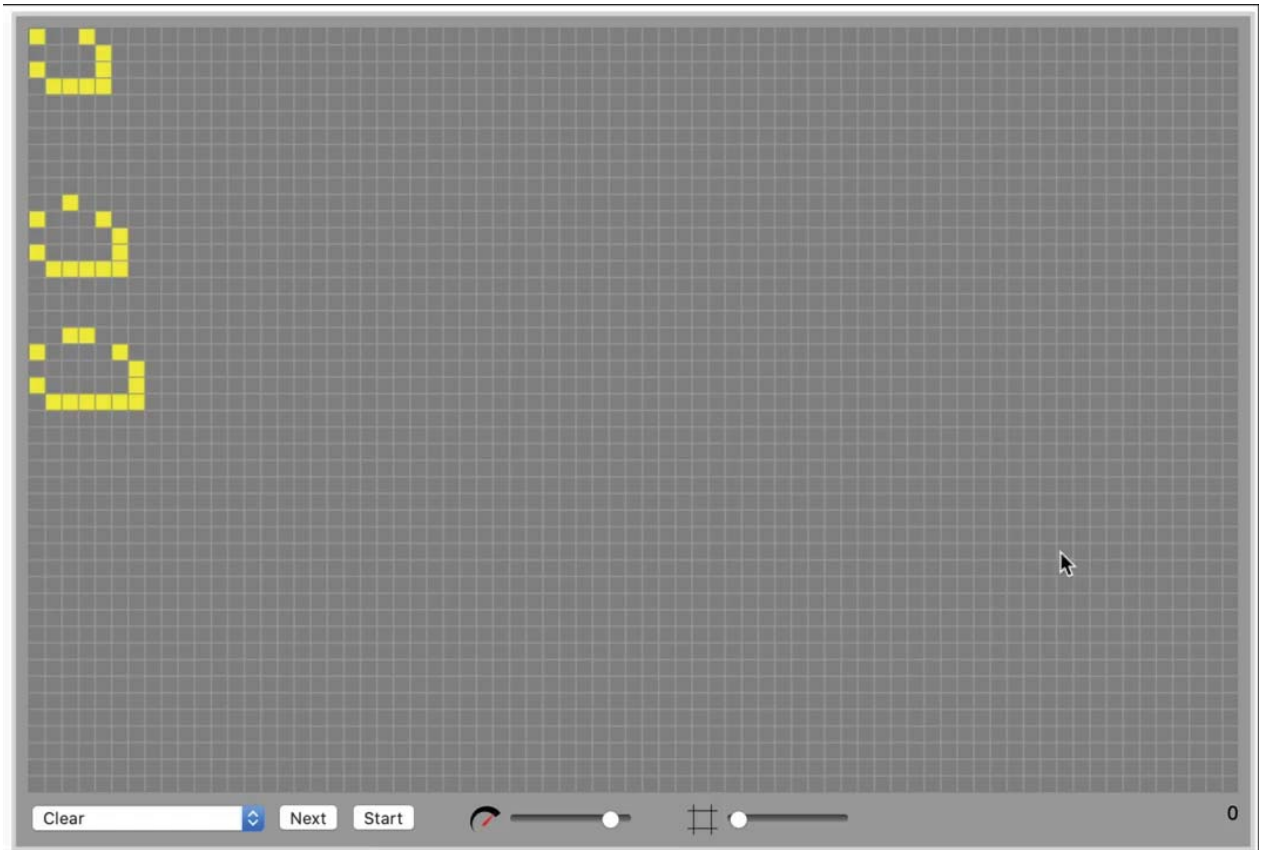
manera oscilatoria. Su prevalencia es consistente con el hecho de que se forman a partir de solo cinco células vivas, ya que las figuras estables más pequeñas son, con mucho, los residuos mas comunes de la evolución a largo plazo (McIntosh, 2010). El planeador más pequeño, denominado justamente *glider*[87] muta, como se observa en el siguiente video, mientras recorre el tablero de manera oblicua.



Video 9: planeador

Otros planeadores utilizan más células y poseen nombres como *nave espacial ligera*, *mediana y pesada.* Estos oscilan entre cuatro estados y avanzan un casillero de manera horizontal cada dos mutaciones.

87 Planeador en inglés



Video 10: planeadores tipo nave espacial ligera, mediana y pesada

Existen muchos otros *gliders* de mayor o menor complejidad.

Otras configuraciones

Muchas otras configuraciones iniciales pueden ser mencionadas. Algunas de ellas se extinguen en pocos pasos, otras se convierten en naturalezas muertas a las pocas iteraciones, otras, tienen destinos similares, pero después de muchos pasos (en algunos casos hasta 5000, por ejemplo). Estas últimas se denominan “Matusalén” y Martin Gardner las define como “aquellas configuraciones de no más de 10 celdas que no se estabilizan hasta pasadas las 50 generaciones” (Gardner, 1971). Otras configuraciones se denominan “trenes a vapor” porque van dejando una estela de células a medida que avanzan y otras

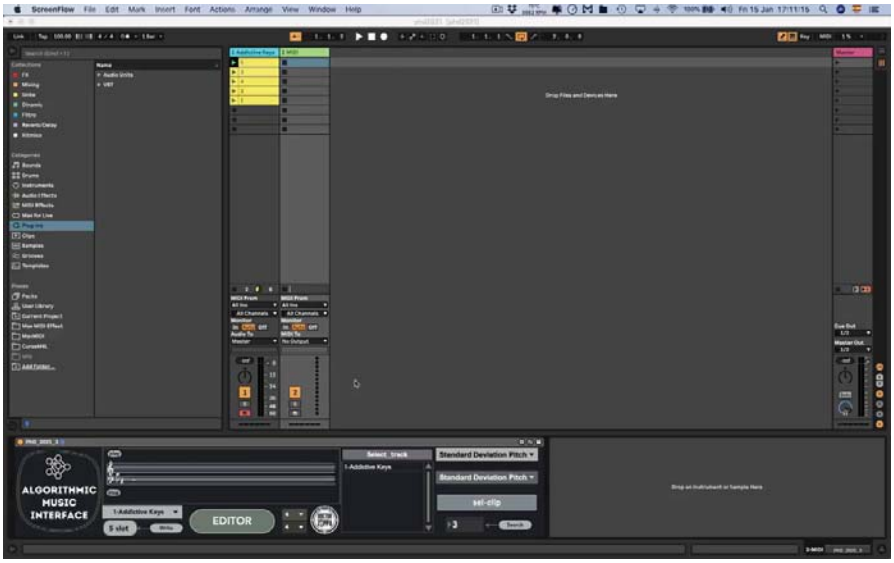
llamadas “cañones” están formadas por un patrón con una parte principal que oscila y que emite naves espaciales periódicamente.

Capítulo 3. Algorithmic Music Interface

3.1 - Introducción

En este capítulo se expondrán los pormenores de cómo se implementaron en *Max for Live* los generadores expuestos en el capítulo 2. Para ello se verán los módulos que se encargan de generar alturas, ritmos y dinámicas incluidos dentro de la herramienta creada para este trabajo y denominada *Algorithmic Music Interface* (*AMI*). Se explicarán los desafíos enfrentados y las soluciones encontradas para lograr implementar los algoritmos descriptos en el capítulo 2. La interfaz de usuario (GUI por sus siglas en inglés) es la siguiente:

Figura 108: interfaz de usuario



Video 11: GUI de los módulos generadores

3.2 - Generador de alturas

Al accionar el botón “EDITOR” se abre una ventana desde donde se generan las alturas, ritmos y dinámicas. Como se puede apreciar en la figura 109, la sección desde donde se pueden generar datos simbólicos dentro de *AMI* se divide en tres módulos: alturas (izquierda-verde), ritmo (centro-azul) y dinámica (derecha-anaranjado). A continuación se observa una imagen de dicha interfaz y el detalle del módulo para generar alturas:

Figura 109: módulo generador de alturas

En este caso, al igual que con los otros dos módulos, la generación de datos se produce en dos pasos, en uno de ellos, el denominado *perfil*, el usuario escoge el algoritmo que utilizará y los parámetros que modificarán el resultado obtenido, y en el otro, *alturas*, el usuario especifica las notas que serán asignadas al resultado del paso anterior. El orden en que se ejecuten estos pasos no tiene importancia, pudiéndose comenzar por uno u otro indistintamente. En la generación de los perfiles es donde se utilizan los algoritmos del capítulo 2, así, por ejemplo, es posible escoger el generador de Random Walk[88] obteniendo así un perfil como el que se observa a continuación:

Figura 110: generación de Random Walk

Luego, las notas seleccionadas en la interfaz serán mapeadas de acuerdo a este perfil y al ámbito que el usuario escoja. Por ejemplo, si se utilizara una escala de Do menor con un ámbito de 3 octavas se obtendría la secuencia que se observa y escucha a continuación:

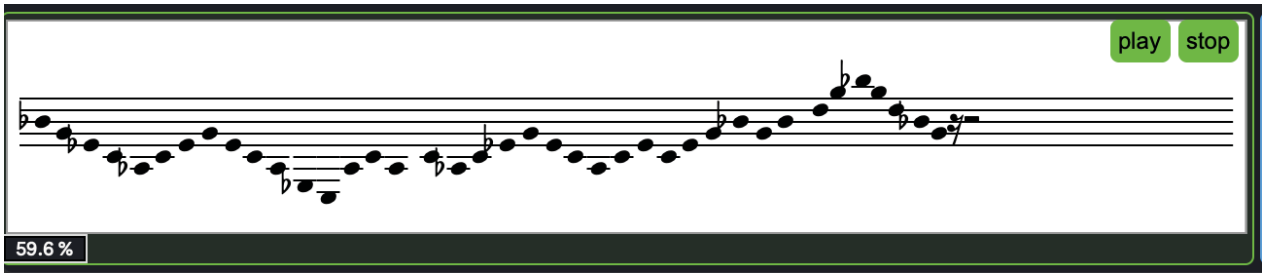
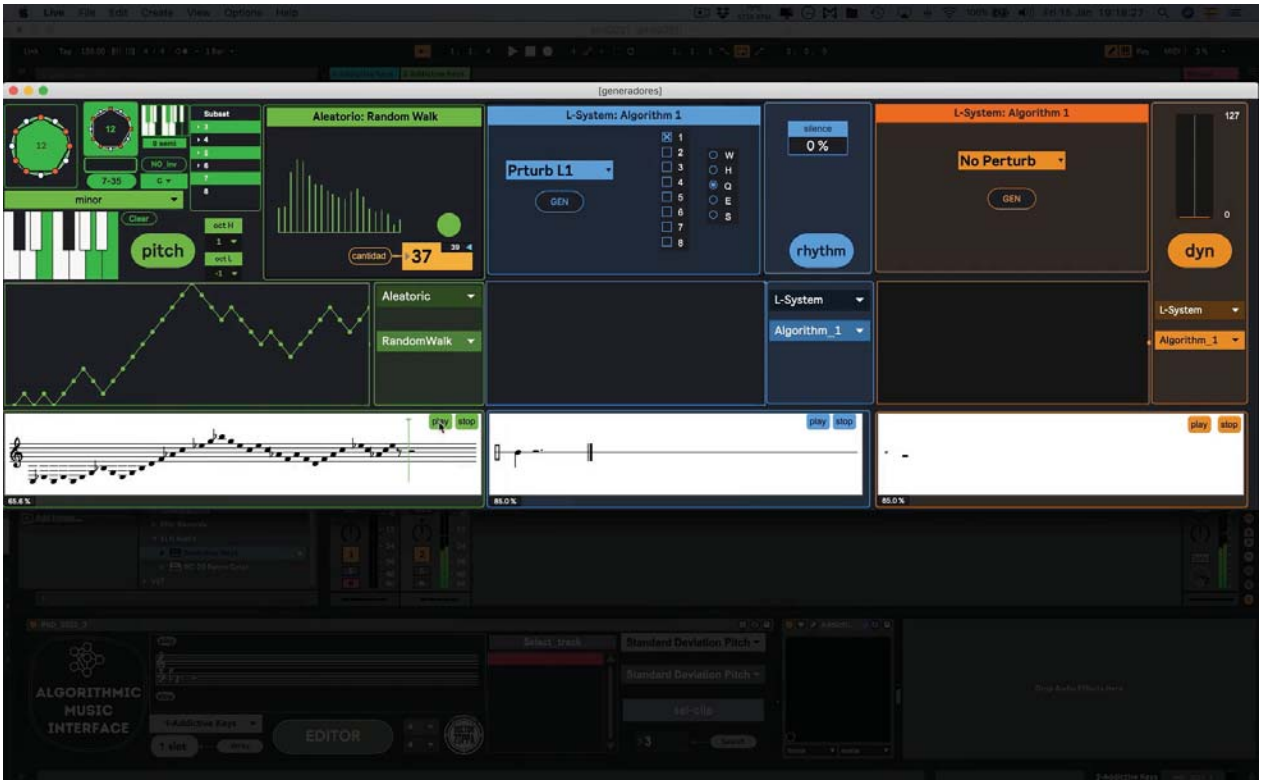


Figura 111: mapeo del perfil generado con un Random Walk

[88] Ver capítulo 2 sección 4.1

Se implementó este sistema (en el que el perfil y las notas se generan de manera “independiente”) para que el usuario pueda fácilmente mapear un perfil dado con diferentes notas o usar uno diferente, es decir modificar el perfil sin cambiar las notas que utilizará y viceversa. En el siguiente ejemplo se muestra cómo, el cambiar uno u otro parámetro en la interfaz no solo no acarrea ningún inconveniente sino que, por el contrario, brinda flexibilidad al usuario.



Video 12: ejemplo de intercambio entre perfil y notas

3.2.1 Módulo para seleccionar y modificar alturas

Interfaz

En este sector del módulo de alturas, el usuario puede escoger las notas que serán utilizadas para ser mapeadas de acuerdo al perfil seleccionado. Existen cinco interfaces

diferentes desde donde escoger estas alturas (señaladas en la figura 112 con círculos numerados):

1) Interfaz de teclado.
2) Menú de escalas.
3) Círculo cromático de alturas.
4) Pitch Class set (PCs).
5) Pitch Class subsets.

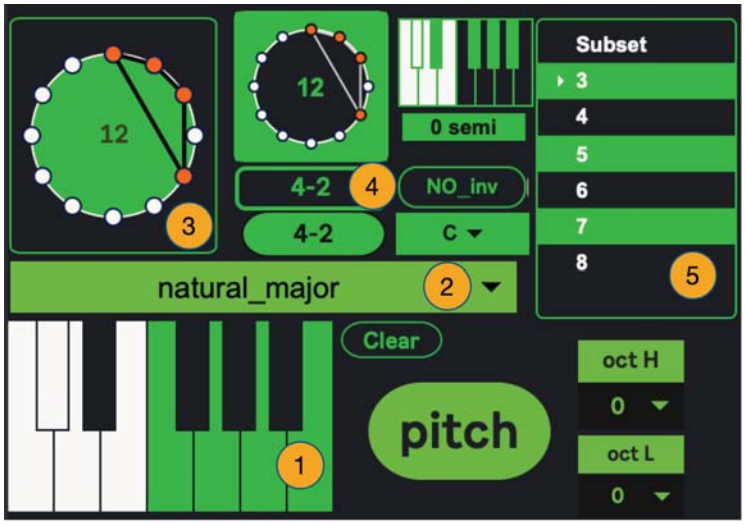


Figura 112: interfaz para seleccionar las alturas

La interfaz de teclado (1) no necesita mayor explicación, desde allí se pueden seleccionar o des-seleccionar las notas deseadas. Desde el menú de escalas (2) se puede escoger una fundamental y luego una de las sesenta y cinco escalas disponibles. El círculo cromático de alturas (3) es un diagrama tipo reloj que permite seleccionar una nota haciendo clic en alguno de los pequeños círculos que se encuentran distribuidos alrededor de la circunferencia principal. El círculo que se encuentra en la posición 0:00 hs representa al Do, el siguiente a la derecha será Do# y así sucesivamente. Esta interfaz permite

representar de manera sencilla las relaciones entre las doce alturas temperadas como así también la relaciones de distancia entre las notas (McCartin, 1998). Para des-seleccionar un punto se debe hacer command + clic (OSX) o control + clic (Windows) sobre el mismo. En (4) es posible ingresar un PCs con el formato *cardinal-set* de acuerdo al orden que le asigna Allen Forte (Forte, 1973). En (5), también tomado de la teoría de los PCs, es posible seleccionar uno de los subgrupos que se encuentran contenidos dentro del conjunto formado por las notas escogidas a través de cualquiera de las interfaces expuestas.

Una vez escogidas las alturas, estas pueden modificarse a través de las interfaces que se observan en la figura 113. De esta manera, a través de (1) se puede estipular el registro que utilizarán las notas para adaptarse al perfil generado, siendo el 0 la octava central y pudiéndose especificar un registro que va desde la octava -3 a la 3 (desde el Do 24 al 108 en valores MIDI). En (2) es posible escoger entre las notas seleccionadas o su versión invertida (por lo tanto, por ejemplo, Do - Mib - Sol se convierte en Do - Fa - La). En (3) es posible transportar las notas seleccionadas. También la interfaz incluye tres espacios donde se pueden observar los cambios introducidos: en (4) muestra el PCs que resulta de las notas escogidas, en (5) se puede ver cómo se modifica el círculo cromático si se modifican las notas a través de la inversión y/o la transposición y, por último, en (6) se puede observar lo mismo que en (5) pero en la interfaz de teclado virtual.

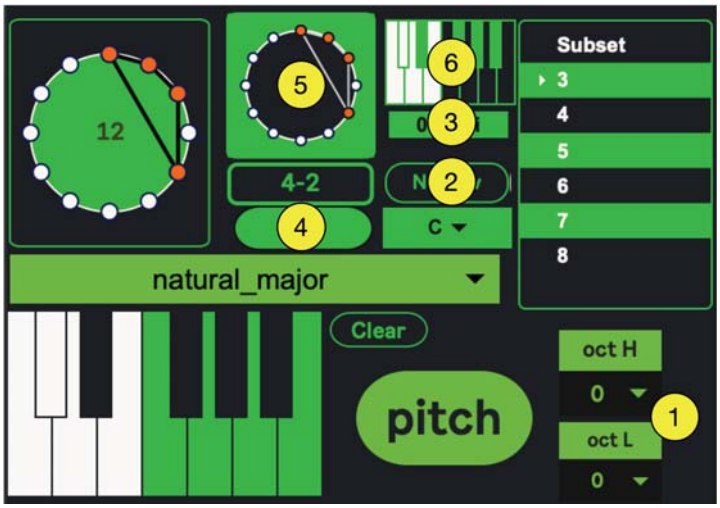


Figura 113: interfaces para modificación y observación

Desafíos de la programación.

En cuanto a la programación de la interfaz expuesta en la figura 109 los desafíos encontrados no fueron muchos. El menú de escalas (figura 112 (2)), por ejemplo, no implica más que utilizar un objeto como **coll** que permite declarar un array de valores que serán, en este caso, los Pitch Class (0 = Do, 1 = Do#, etc.) correspondientes a la escala declarada en el menú, así, por ejemplo, la escala mayor será igual a [0 2 4 5 7 9 11]. Sin embargo los dos problemas que implicaron un poco más de programación fueron, la posibilidad de ingresar los PCs a través de la interfaz de teclado virtual (llamada **kslider** en *Max*) y la posibilidad de obtener los subsets que se encuentran contenidos en dicho PCs.

En el primer caso, el problema es que el objeto **kslider** envía sus valores de manera secuencial, esto es, como números individuales que corresponden a la nota MIDI seleccionada, pero la programación necesita que estos valores salgan como un PCs, es decir, como un grupo de números que representan el acorde seleccionado y que, además,

al hacer clic sobre una tecla seleccionada, esta se des-seleccione eliminando ese valor de la lista resultante. En el siguiente video se puede observar cómo es que **kslider** envía sus valores y en en la figura 114 se observa la programación.

Video 13: funcionamiento de **kslider**

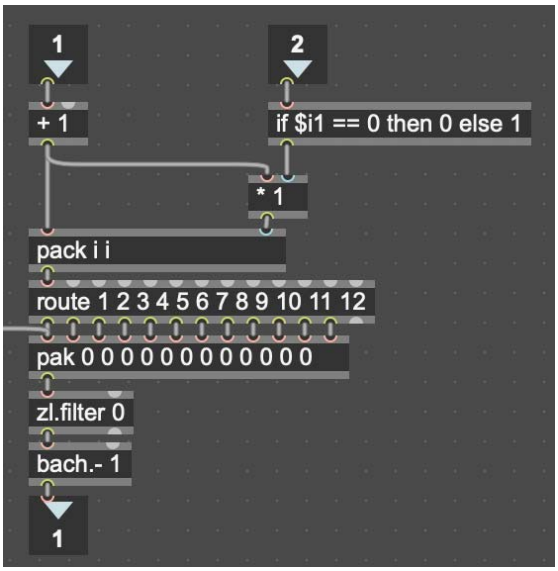
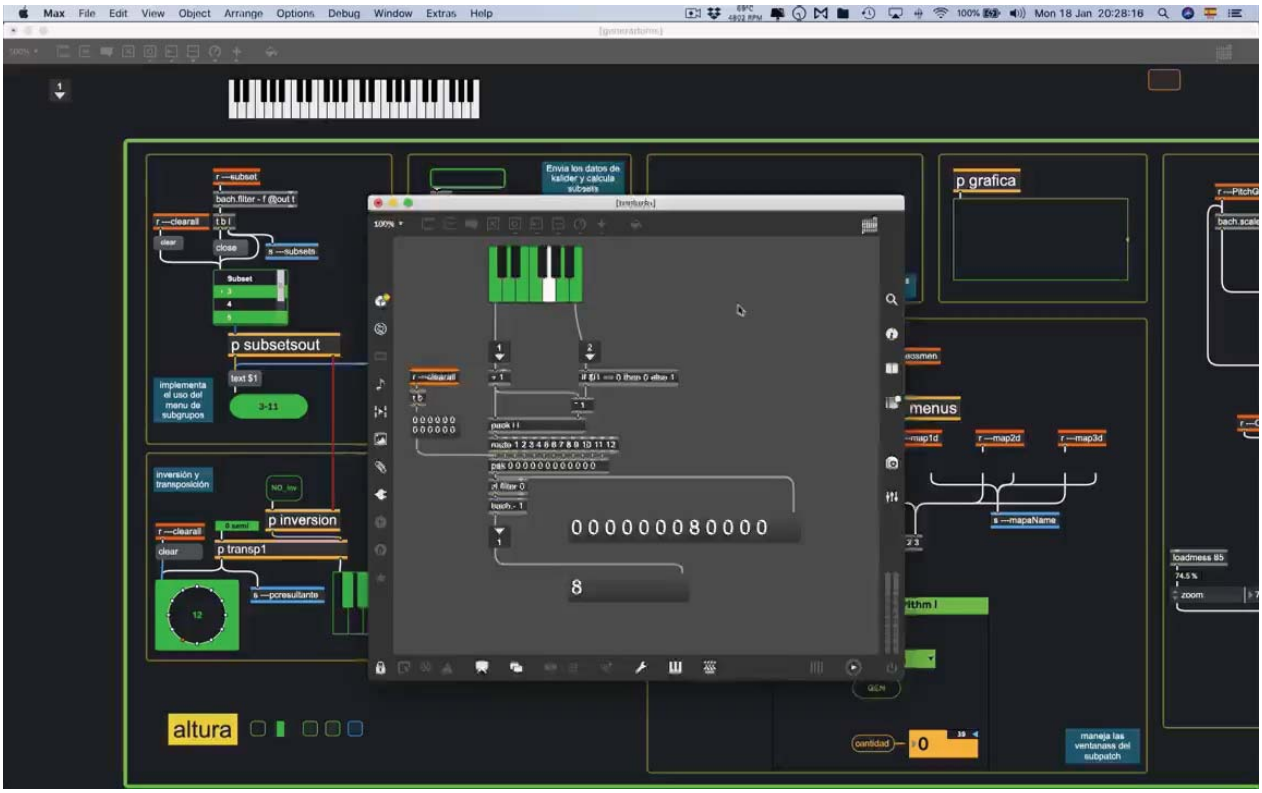


Figura 114: solución al problema de **kslider**

Por las entradas 1 y 2, que se ven en la parte superior del patch, ingresan las alturas (0 - 11) y la velocity (127 para la nota activada y 0 para la desactivada) respectivamente. Las alturas son sumadas más uno (+ 1) para que el rango vaya desde 1 a 12; por otro lado, las velocities atraviesan el condicional **if $i1 == 0 then 0 else 1** para que, si el valor que ingresa es igual a cero se mantenga, pero cualquier otro valor que ingrese será igual a uno. Del objeto **+ 1** salen dos cables: uno va hacia un multiplicador *** 1** y el otro va a un objeto **pack i i.** De esta manera, entonces, si se activa una nota en **kslider**, y esa nota es un Sol, por ejemplo, esta tendrá un valor de 7, que al sumarse **+ 1** será igual a 8 y este valor ingresa al objeto que lo multiplicará por 1 (ya que, como la nota ha sido activada, envió una velocity de 127 que se convierte en un 1 al atravesar el objeto **if**). Así, entonces, el 8 sale sin cambios. El objeto **pack i i**, crea una lista que consta del valor que ingresa por su entrada izquierda (en este caso un 8) y el que ingresa por la entrada derecha (también un 8). Así se obtiene una salida igual a 8 8. Dicha lista ingresa al objeto **route** que permitirá el paso de un dato siempre que este se encuentre antecedido por el valor declarado en sus argumentos. En este caso, como el objeto **route** especifica que dejará pasar cualquier valor que se encuentre antecedido por 1, por 2, por 3 y así hasta 12, entonces al recibir el **8 8** deja pasar el segundo 8 a través de su octava salida. Este valor es enviando luego a **pak 0 0 0 0 0 0 0 0 0 0 0 0** que creará una lista que contendrá el valor que ingresa por alguna de sus entradas y rellenará el resto con ceros, es decir que en este caso se obtendrá un [0 0 0 0 0 0 0 **8** 0 0 0 0] pero el objeto siguiente, es decir, **zl.filter 0** filtrará los ceros dejando pasar solo el 8. A continuación, si se ingresa un Fa desde **kslider**, se obtiene la siguiente lista: [0 0 0 0 0 **6** 0 **8** 0 0 0 0] que al atravesar el filtro quedará **6 8**. Pero, si a continuación se des-selecciona el 8, entonces su velocity será ahora de cero que al juntarse en **pack i i** quedaría **8 0** y al atravesar el **route** quedará solo 0 que será filtrado por **zl.filter 0,** así queda, solo el 6.



Video 14: resultado obtenido de la programación expuesta.

La programación necesaria para obtener los subgrupos contenidos en el PCs escogido requiere de mayor cantidad de objetos. En la figura 115 se muestra esta programación. Aquí hay dos “ramas”, una, a la izquierda, formada por los objetos **bach.primeform**, **regexp**, **coll** y **route** y la otra que es la que se despliega a la derecha. La primera sirve para mostrar, en la interfaz de generación de alturas, la nomenclatura de Forte del PCs seleccionado. Para esto primero convierte al PCs que ingresa por el **inlet 1** (1) en su forma prima, esto significa que escoge primero el orden normal del grupo de alturas y luego lo ordena de manera ascendente comenzando desde 0 (Forte, 1973). Este proceso se lleva a cabo dentro del objeto **bach.primeform**. Luego, en **regexp**, se remueven los espacios existentes entre los valores obtenidos. De esta manera, [0 2 4 6], por ejemplo, se convertirá en 0246 y como el 0 es redundante, el sistema lo elimina automáticamente quedando 246. Dicho valor ingresa al array **coll** donde se encuentra almacenada una lista

de valores con su correspondiente nomenclatura de Forte. Así 246 será “4-21”.

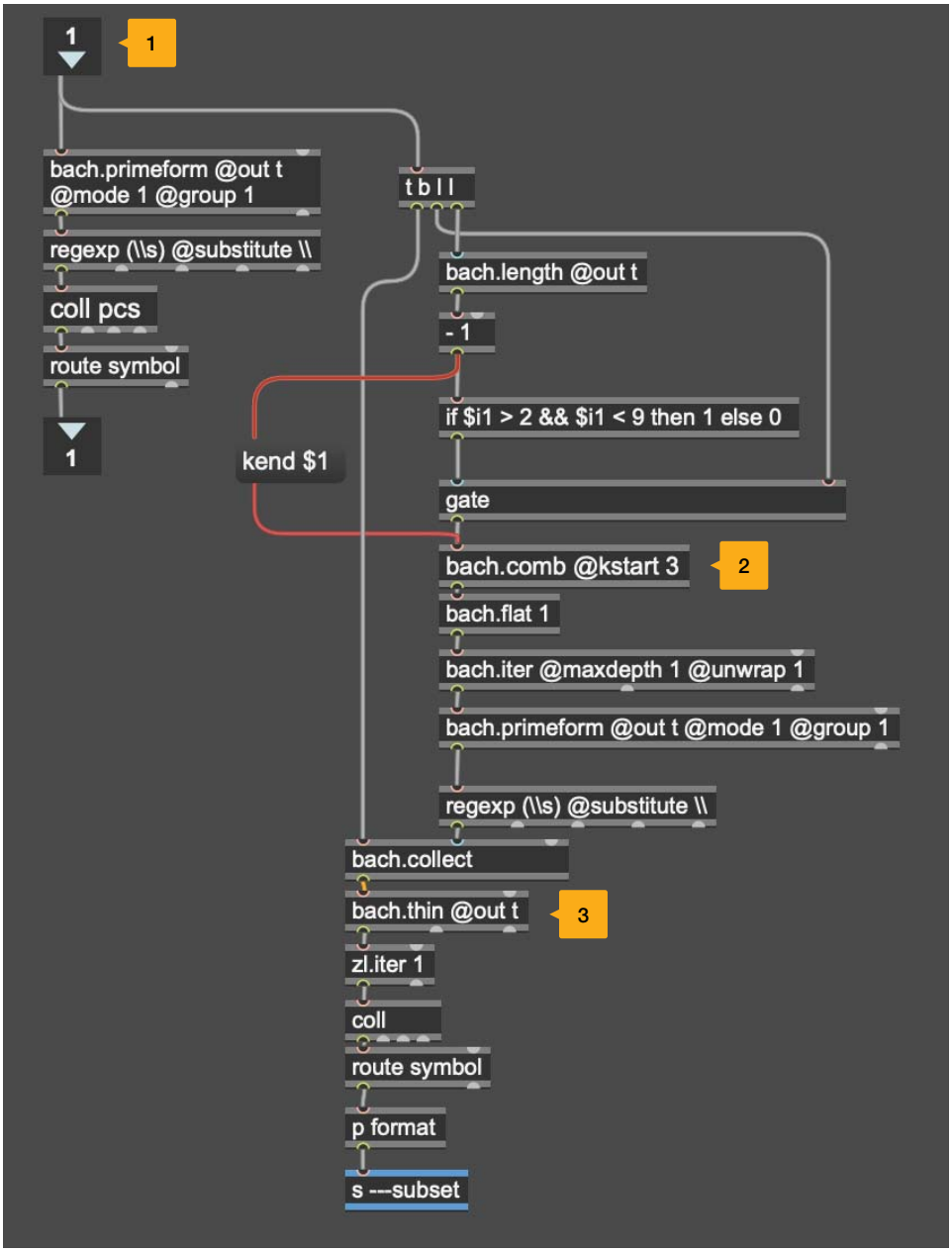


Figura 115: programación para obtener los subgrupos

La segunda “rama” es la encargada de calcular y mostrar los subgrupos contenidos dentro del PCs seleccionado. Para ello, lo primero que realiza la programación es limitar el paso de PCs para que solo se calculen los subgrupos de aquellos con una cardinalidad mayor a 2 (ya que los subgrupos de un grupo de 2 notas serían notas aisladas) y menor a 9. La

decisión de limitar los PCs para que solo sean calculados aquellos con una cardinalidad máxima de 8 se debe a dos razones: la primera es que la cantidad de subgrupos contenidos en una lista de *n* valores es igual a 2^n es decir que en el caso de 12 valores tendríamos 4096 subgrupos. Esto implica un costo computacional innecesario para los fines buscados en esta programación. La segunda, también ligada a esta última razón, es que, a mayor cardinalidad del grupo, menor cantidad de subgrupos diferentes se obtienen. Así, por ejemplo en el caso extremo de tener un PCs que contenga todas las notas de la escala cromática dentro de una octava: [0 1 2 3 4 5 6 7 8 9 10 11], los subgrupos resultantes serían todas las combinaciones posibles de notas (desde 3 a 12) que pueden obtenerse con este conjunto. Es decir que este conjunto contendría a todos los subgrupos. Por ello se limitó estos subgrupos a, como máximo, ocho elementos.

El objeto **bach.comb** (2) es el encargado de calcular todos los subgrupos contenidos en el PCs, desde los de cardinalidad 3 (**@kstart 3**) hasta los que poseen una cardinalidad igual al tamaño del PCs menos 1. Es decir que por ejemplo para el PCs [0 1 5 7 8 9] se calcularán los subgrupos que contengan 3, 4 y 5 elementos. Luego estas listas de alturas son iteradas (es decir, enviadas de a una) y una vez calculada su forma prima se les suprime los espacios como se hizo en la rama de la izquierda, luego vuelven a juntarse y atraviesan el objeto **bach.thin** (3). Esto permite eliminar cualquier subgrupo duplicado. Finalmente el valor resultante ingresa a **coll** para obtener la nomenclatura del PCs correspondiente.

3.2.2 Generador de perfiles

La interfaz para generar perfiles consta de tres partes. En (1) es posible escoger el

algoritmo deseado a partir de su categoría (L-System, caos, aleatorio, genético y autómata celular). Las posibilidades específicas son las siguientes:

- L-System: algoritmo 1 y algoritmo 2.
- Caos: mapa 1D (logístico, congruencial, sine, CUSP, cuadrático y gauss), mapa 2D (henon, henon phase, clifford, jong, gingerbread, burning ship e ikeda) y mapa 3D (lorenz).
- Aleatorios: random walk, lineal, triangular, exponencial y bilateral
- Genéticos
- Autómata celular

En (2) es posible modificar los parámetros correspondientes a cada algoritmo y en (3) observar un gráfico del perfil generado.

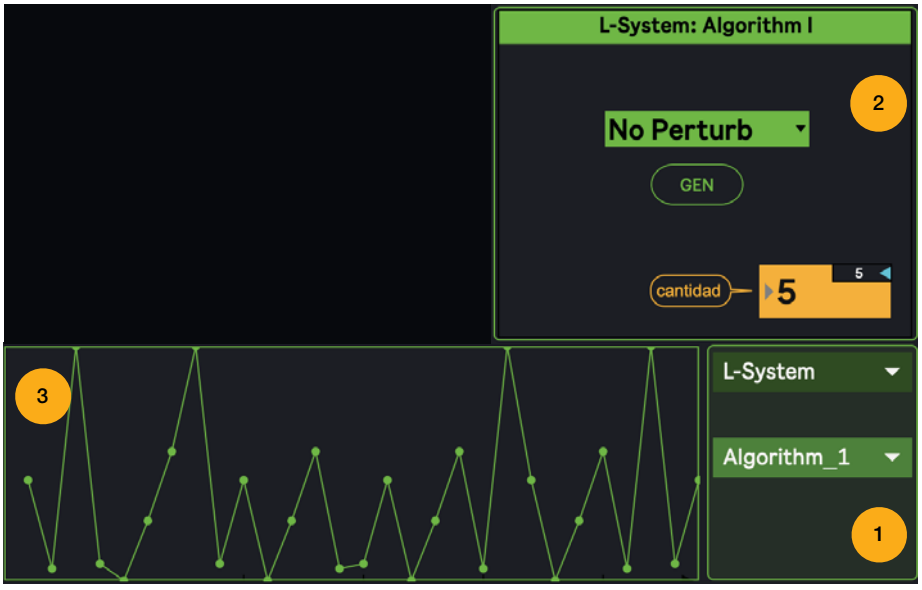


Figura 116: interfaz para la generación de perfiles

L-System

Esta implementación posee dos variantes. En la primera se ha buscado crear un mecanismo en el cual las reglas con las que se alimenta el sistema se generen de manera automática para que el usuario no deba lidiar con estos datos. El objeto encargado de generar los *L-Systems* aquí es **cage.chain**, a este objeto debe ingresarse por el tercer **inlet** las reglas deseadas $P = \alpha \rightarrow \sigma$ con el formato [[α] σ]. Entonces, por ejemplo $P = A \rightarrow BCAB$ y $B \rightarrow CACB$ se escribirá [[A] B C A B][[B]C A C B]. Por el segundo **inlet** se ingresa el axioma ω (en este caso A) y por el primer **inlet** un entero igual a la cantidad de iteraciones deseadas (en este caso 4). Ver la siguiente figura:

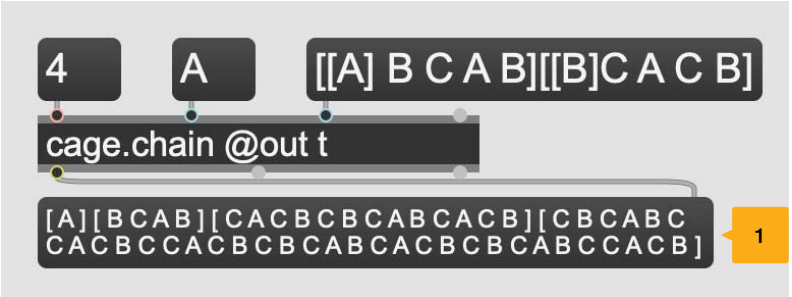


Figura 117: **cage.chain**

El resultado que se ve en (1) muestra que se generó primero el axioma (A), luego las letras asociadas a este ([B C A B]) y luego una cadena de símbolos resultantes de aplicar las reglas estipuladas a la cadena [B C A B]

$B \rightarrow$ [C A C B]

$C \rightarrow C$

$A \rightarrow$ [B C A B]

$B \rightarrow$ [C A C B]

Quedando entonces la cadena [C A C B C B C A B C A C B]. Luego, por último, se aplica el mismo procedimiento sobre esta última cadena ya que la cantidad de iteraciones especificadas son 4.

Para alimentar este algoritmo de manera automática se ha implementado un sistema que genera siete valores aleatorios sin repetición entre 0 y 127[89].

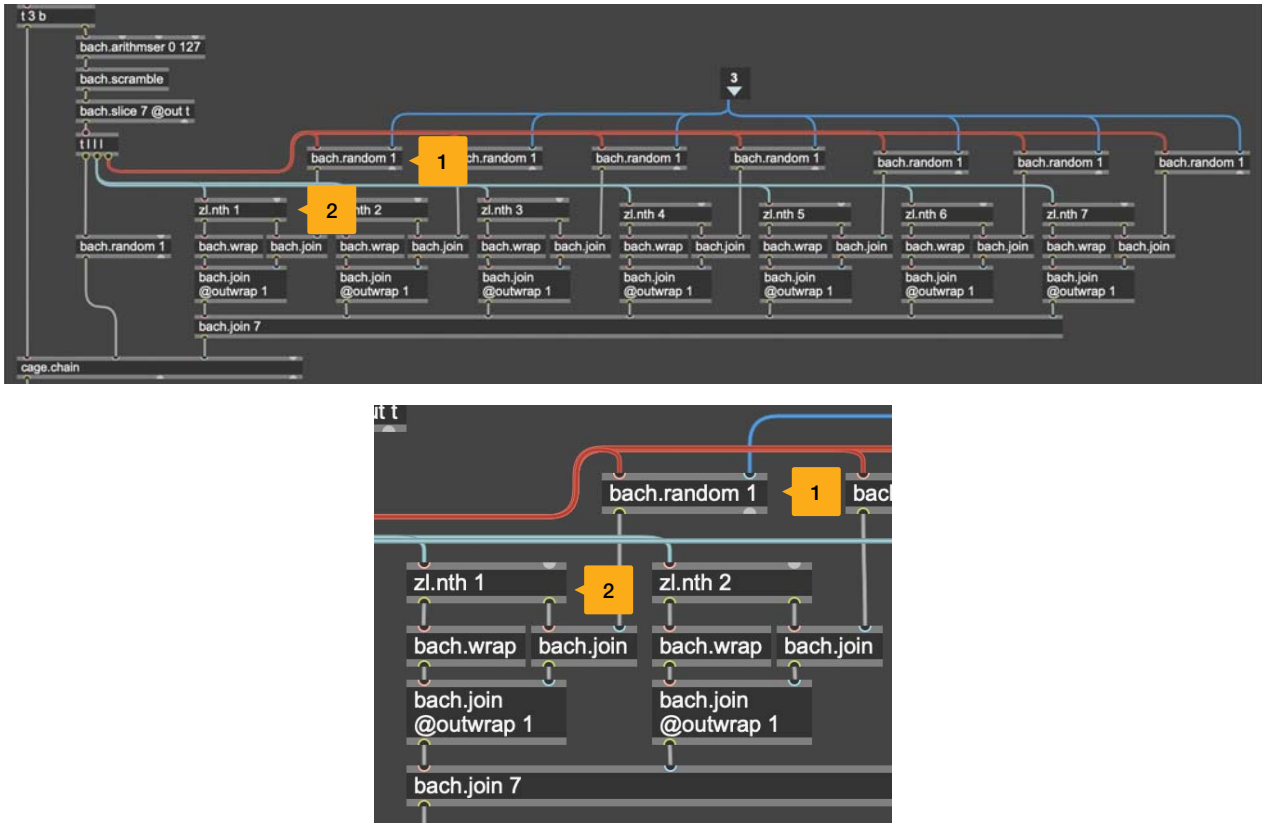


Figura 118: generación automática de L-System.

Así, estos siete valores ingresan, primero, a siete objetos **bach.random** (1) que escogerán uno de ellos al azar (aquí sí puede suceder que se escoja el mismo número más de una vez). Entonces si la primera lista generada fuera por ejemplo [71 78 122 118 92 19 11]

89 La elección es arbitraria. No responde a los 128 valores que se utilizan habitualmente en MIDI.

cada objeto **bach.random** podría escoger, por ejemplo, los siguientes valores: 1) 71, 2) 92, 3) 78, 4) 122, 5) 11, 6) 11 (aquí se repite) y 7) 11 (se repite nuevamente). Luego esta misma lista inicial [71 78 122 118 92 19 11] ingresa a sendos objetos **zl.nth 1**, **zl.nth 2**, …, **zl.nth 7**. Así el primer **zl.nth** escoge el primer valor de la lista (71) y el segundo el segundo valor (78) y así sucesivamente hasta el último (11). Estos valores salen por el **outlet** izquierdo de **zl.nth** mientras que por la salida derecha saldrá el resto de los números de la lista. A estos valores remanentes se les acoplan sendos enteros generados por los **bach.random**. Entonces, por ejemplo, en el primer caso se obtiene por la salida izquierda de **zl.nth 1** el valor 71 mientras que por la derecha saldrá [78 122 118 92 19 11] pero a esta lista se le acoplará el valor escogido por **bach.random** que es también un 71; entonces nos quedará [78 122 118 92 19 11 71]. Esta lista se juntará con el número que sale por la primera salida de **zl.nth 1** pero este valor será puesto entre corchetes [ ], gracias al objeto **bach.wrap**, mientras que al resultado final también se le agregarán corchetes gracias al atributo **@outwrap 1** que posee el objeto **bach.join**. Así el resultado final será [[71] 78 122 118 92 19 11 71]. Este mismo proceso se repetirá para cada **zl.nth** y todos se juntarán en el objeto **bach.join 7** por lo que se obtendrá un resultado igual a: [[71] 78 122 118 92 19 11 71] [[78] 71 122 118 92 19 11 92] [[122] 71 78 118 92 19 11 78] [[118] 71 78 122 92 19 11 122] [[92] 71 78 122 118 19 11 11] [[19] 71 78 122 118 92 11 11] [[11] 71 78 122 118 92 19 11] que será la regla que alimentará a **cage.chain**. En este caso particular, si el valor inicial fuera 78, entonces el resultado final generado por **cage.chain** será: [78] [71 122 118 92 19 11 92] [78 122 118 92 19 11 71 71 78 118 92 19 11 78 71 78 122 92 19 11 122 71 78 122 118 19 11 11 71 78 122 118 92 11 11 71 78 122 118 92 19 11 71 78 122 118 19 11 11] y el gráfico de este patrón será:

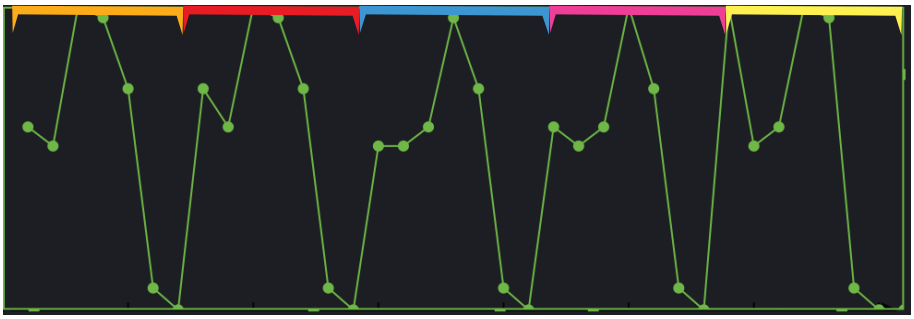


Figura 119: patrón resultate

En la figura 119 es posible observar y escuchar cómo, con el procedimiento explicado hasta aquí, se han obtenido cinco patrones autosimilares que es lo que se espera de un sistema de *L-System*.

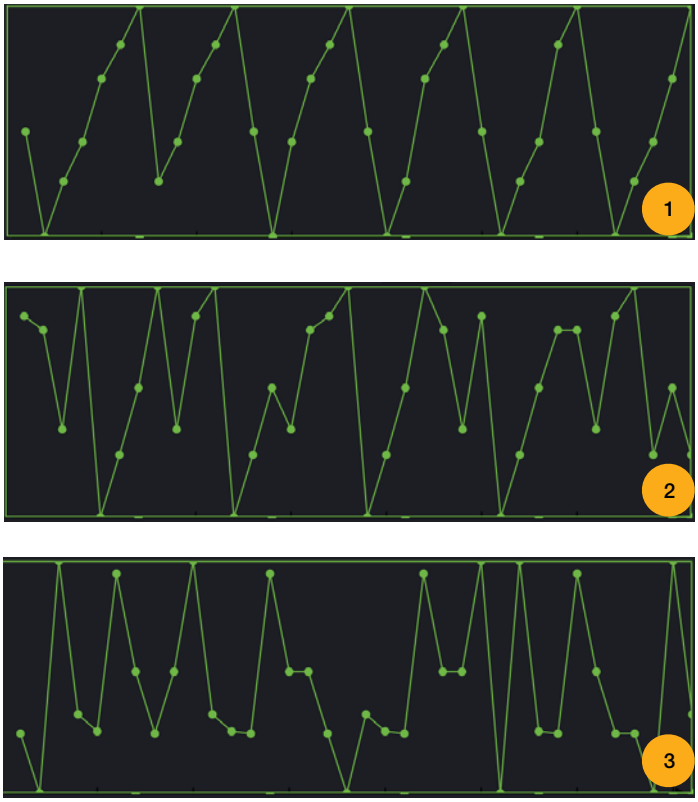


Figura 120: gráficos de las tres opciones de perturbación

Las opciones que nos brinda este generador son las de “No perturb” (1), “Perturb L1” (2) y “Perturb L2” (3). En la primera los objetos **bach.random** no se ponen en funcionamiento, por lo tanto los ciclos no poseen perturbaciones en su forma (sólo modificaciones en el orden de sus valores), en la segunda opción generarán un valor aleatorio y el la tercera dos valores aleatorios, es decir, agregando mayor perturbación a los ciclos. En la figura 120 se observan estas tres opciones compradas.

La segunda variante de *L-System* no es automática como la anterior sino que el usuario puede definir la suya a partir de un conjunto restringido de símbolos. La interfaz creada para tal fin permite escoger un axioma que puede ser una letra desde A hasta K y luego se puede crear la regla a partir de tres filas de cinco símbolos (también desde A hasta K).

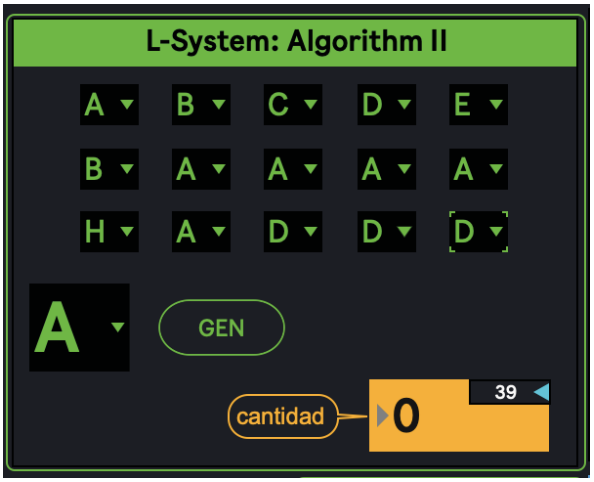


Figura 121: interfaz para el segundo algoritmo de L-Systems

La primer letra de cada fila será el predecesor de la regla y el resto de los símbolos conformarán el sucesor. En el caso de la figura 121 la regla será:

A → [B C D E]

B → [A A A A]

H → [A D D D]

Mientras que el axioma será A.

En esta interfaz, y en todas las que se expondrán para la generación de alturas, ritmos y dinámicas, se podrán especificar la cantidad de notas que se desean generar a partir del número en amarillo señalado con la palabra “cantidad”. El valor del entero que se encuentra sobre este objeto (en negro con un triángulo azul) indica la cantidad máxima de notas que puede generar una algoritmo determinado (en este caso 39) tomando en consideración las posibilidades de generación de los tres algoritmos correspondientes a la altura, al ritmo y a la dinámica. El botón “GEN” será el encargado de poner al algoritmo en funcionamiento para así generar el perfil. A continuación otro ejemplo, si se especifica el siguiente algoritmo:

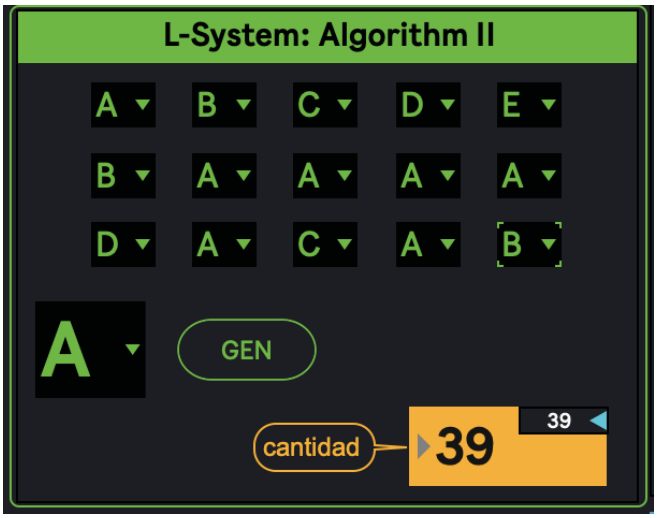


Figura 122: ejemplo de L-System

El resultado que se obtendrá será:

A B C D E A A A A C A C A B E B C D E B C D E B C D E B C D E C B C D E C B C

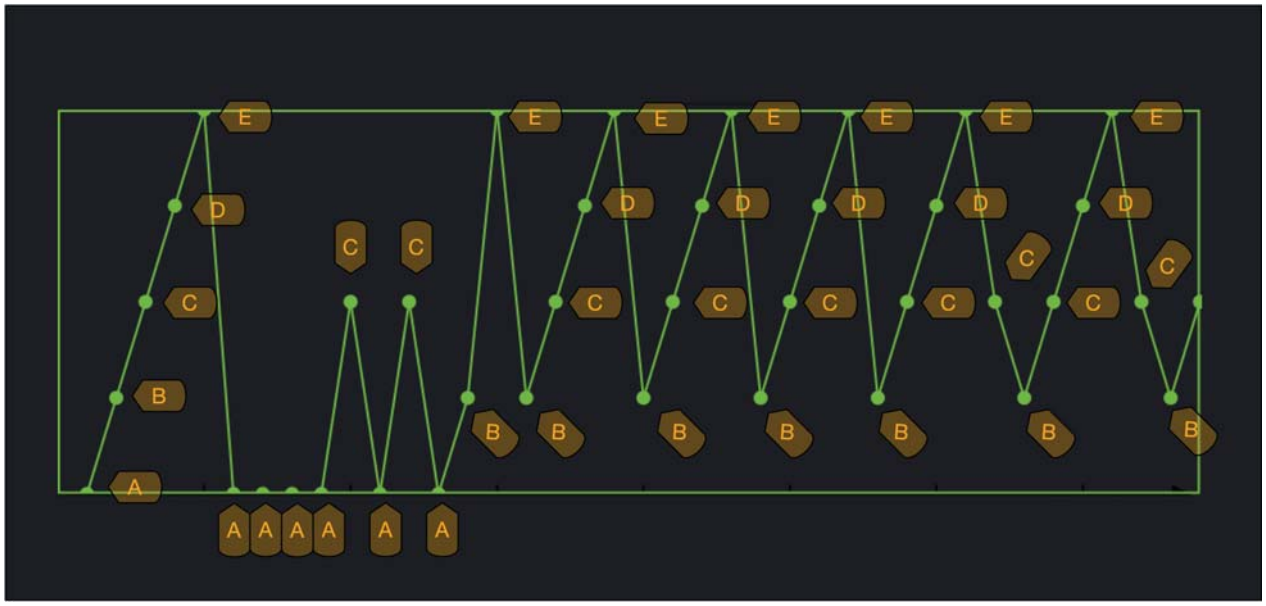


Figura 123: gráfico del sistema generado

Caos

Los algoritmos caóticos, como ya se explicó en el capítulo 2, utilizan funciones recursivas. Para poder implementarlas dentro de *Max* es necesario utilizar un objeto especial llamado **gen**[90]. Este objeto es de suma importancia en el entorno de *Max* ya que, junto con su contraparte para audio **gen~**, permiten crear programaciones de más bajo nivel donde es posible utilizar recursiones que de otra manera serían imposibles o muy complejas de implementar. Al hacer doble clic en **gen** se abre una ventana desde donde es posible editarlo utilizando objetos similares a los de *Max* pero que, como ya se mencionó,

90 De otra manera el software arrojaría un mensaje de error diciendo que el sistema colapsó por saturación (stack overflow)

trabajan a más bajo nivel. Se verá sucintamente cómo se implementaron los algoritmos para generar los mapas caóticos expuestos en el capítulo anterior.

Logístico

Recordar que la fórmula de esta función es:

$$x_{n+1} = r x_n \left(1 - x_n\right)$$

Ecuación 29

La figura 124 muestra la programación que se encuentra dentro del objeto **gen**. Allí se implementa $(1 - x_n)$ utilizando el objeto **!- 1** que sustrae un valor a otro de manera inversa. A este objeto ingresa el valor proveniente de **history x 0.1**, esto produce la recursión enviando por su salida el valor de x_{n-1}, de esta manera se obtiene a la salida de **!- 1** el valor de $(1 - x_{n-1})$. Luego el valor obtenido aquí es multiplicado por x_{n-1} y luego por el valor que ingresa por el **in2** que es el valor de *r*. Así queda $x = r x_{n-1} \left(1 - x_{n-1}\right)$ que es lo mismo que decir $x_{n+1} = r x_n \left(1 - x_n\right)$.

Este algoritmo genera patrones que se repiten en ciclos de 2, 4, 8 notas cuando *r* tiene un valor mayor a 3 y menor a aproximadamente 3.6. En esta implementación los parámetros de los algoritmos están normalizados entre los valores 0. y 1. Es posible ver y escuchar en la figura 125 que el algoritmo empieza a bifurcarse alrededor de 0.17.

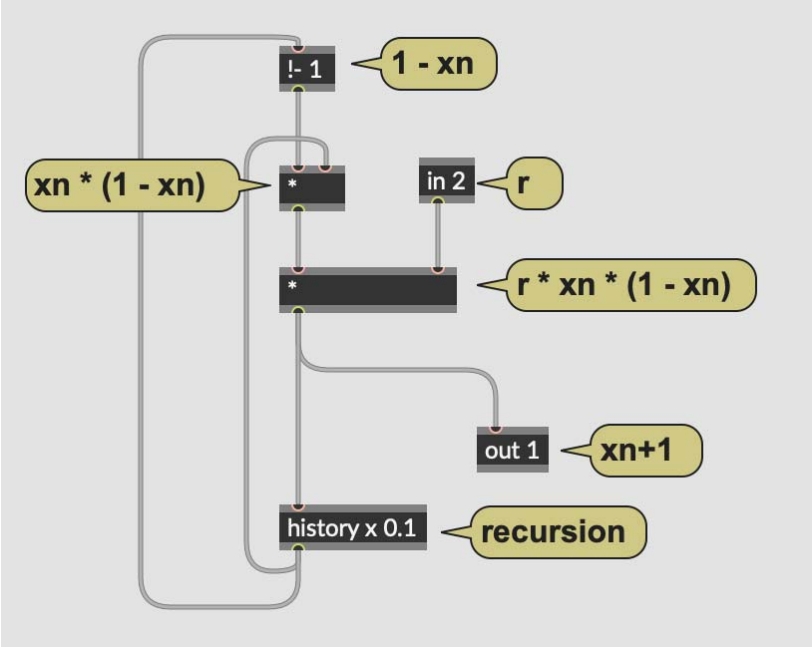


Figura 124: algoritmo implementado en *Max*

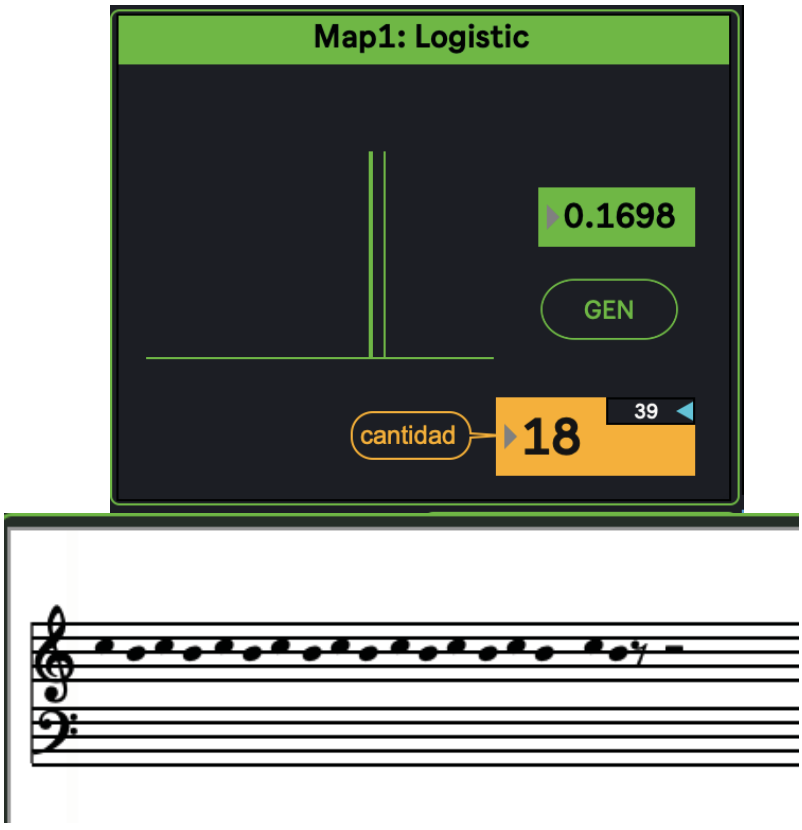


Figura 125: caos logístico r = 0.1698

Con valores de *r* mayores se obtienen ciclos más complejos. Cuando *r* = 1 el comportamiento es estrictamente caótico. En el siguiente video se aprecia el resultado de utilizar diferentes valores de *r*.

Video 15: caos logístico con diferentes valores de r

Congruencial

En la figura 126 se observa la programación que es sencilla de comprender. Se ve que aquí se incorporó un objeto **abs** antes de la salida de la función para obtener siempre valores positivos. Este tipo de algoritmo genera patrones regulares que, mapeados en notas, resultan muy interesantes. Ver en la figura 47 del capítulo 2 cómo evoluciona su salida a medida que se modifican sus parámetros (en ese ejemplo concreto el valor de *a*).

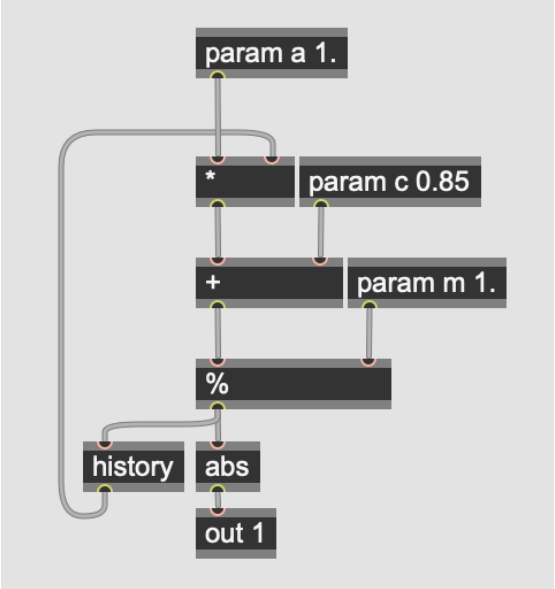


Figura 126: implementación de caos congruencial

Video 16: caos congruencial con diferentes valores

Sine

Como se puede apreciar, la programación de este algoritmo es muy sencilla. El valor constante π ingresa al multiplicador que alimenta a la función seno. A través de **in 2** ingresa el valor de *r* que modifica la amplitud de la función seno.

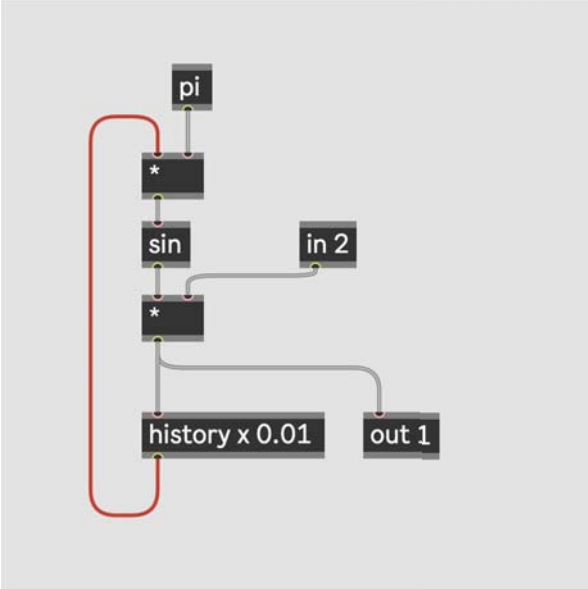
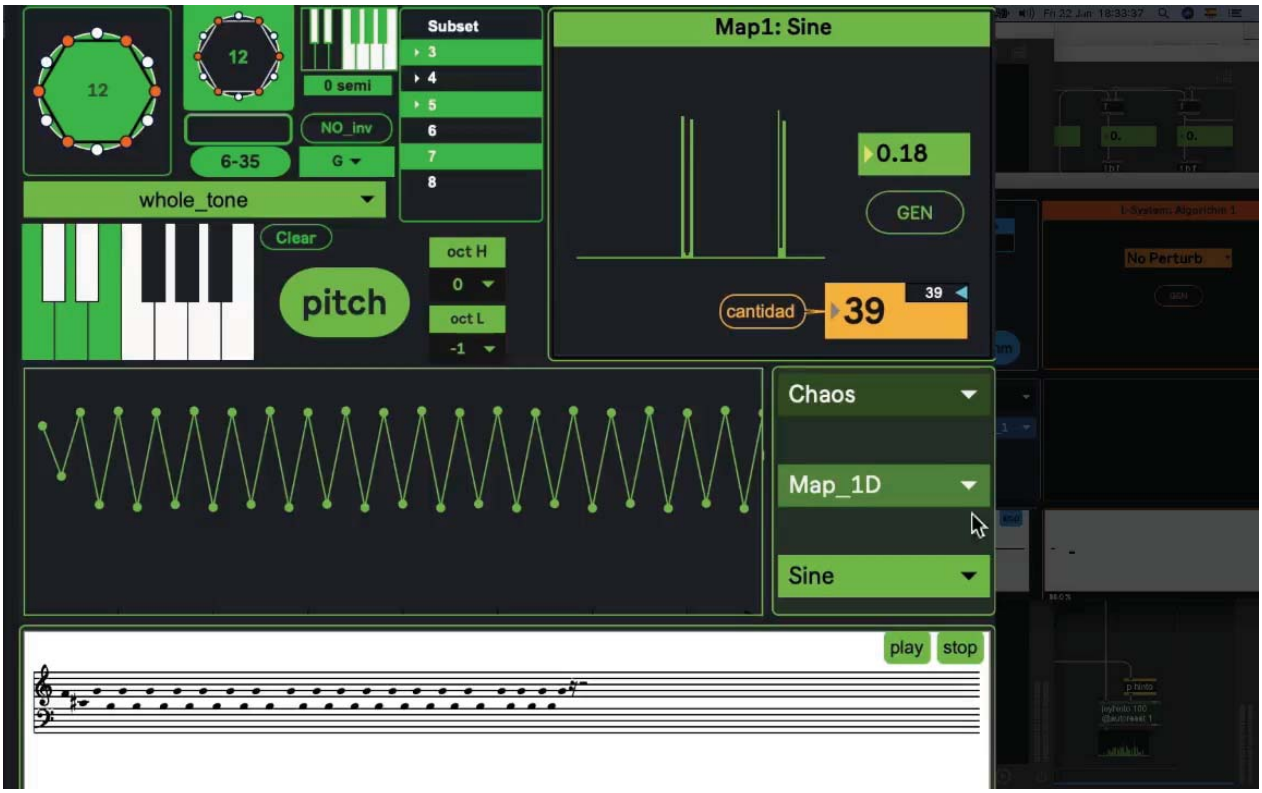


Figura 127: implementación de caos sine

Este algoritmo también comienza con una bifurcación simple cuando el valor de *r* es el mínimo (en este caso 0.). Alrededor de *r* = 0.18 se divide en cuatro bifurcaciones y a partir de *r* = 0.33 comienza a volverse caótico.

En el siguiente video se aprecia un ejemplo utilizando una escala de tonos enteros dentro de un ámbito de dos octavas. Se puede observar como ciertos valores de *r* provocan bifurcaciones interesantes mientras que otros se vuelven más caóticos.



Video 17: caos sine con diferentes valores de r

CUSP

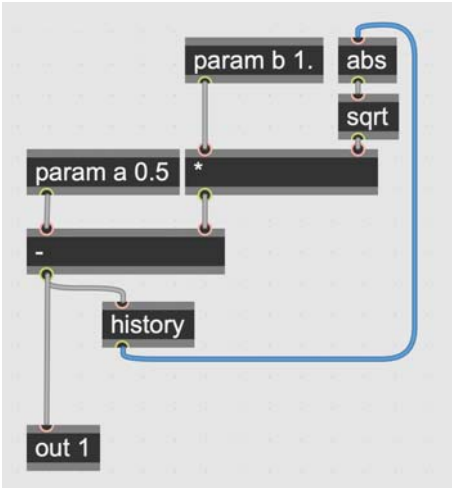
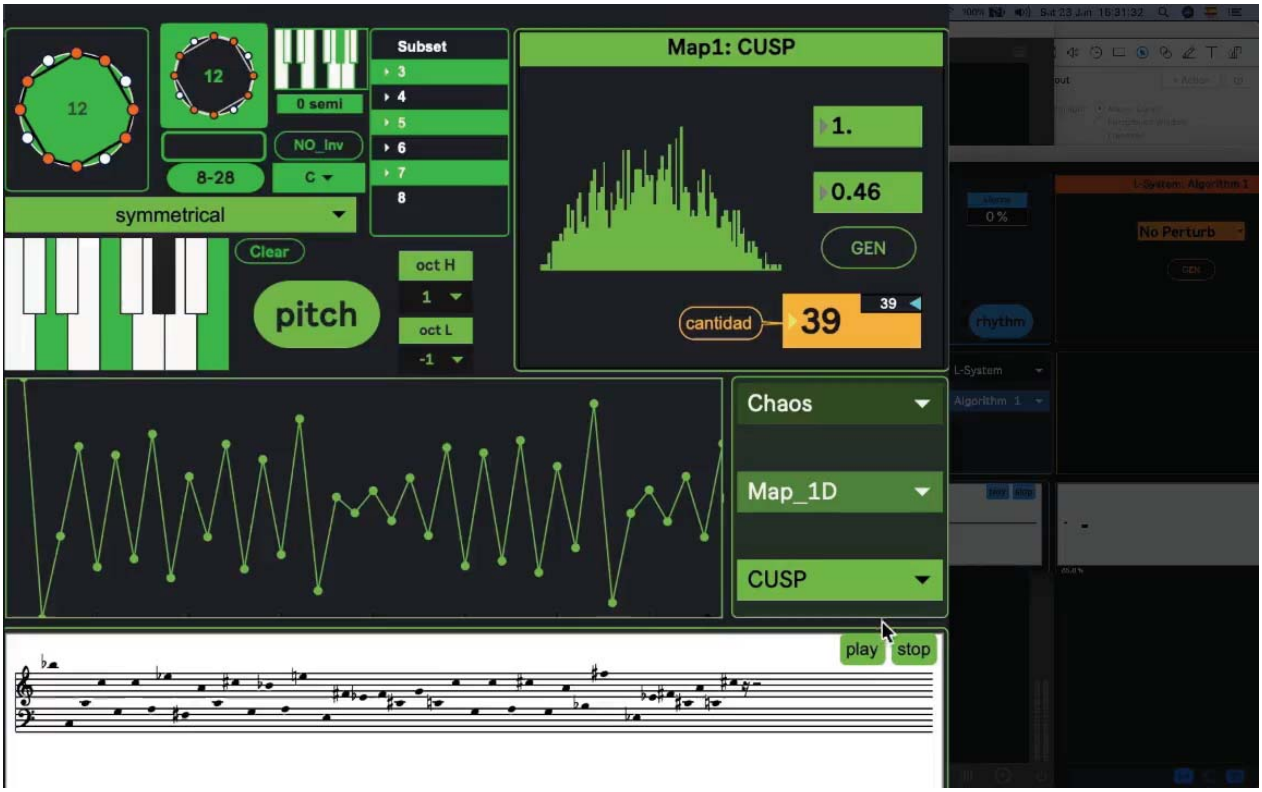


Figura 128: implementación de caos CUSP

En el caso concreto de algunas de las programaciones, como CUSP, es posible comprender su implementación sin mayor explicación que la que puede observarse en la figura 128. Resulta interesante que este algoritmo genera, con ciertos valores de *a* y de *b* (por ejemplo *a = 1 y b* = 0.46), un perfil similar al que genera una aleatoriedad de tipo Gauss, sin embargo con otros valores como *a = 1* y *b = 0.1* se generan dos zonas aleatorias separadas. Esto puede observarse con claridad en el histograma de la interfaz.



Video 18: ejemplos de caos CUSP

Henon

Este es un mapa de dos dimensiones y su fórmula posee dos recursiones: x_n , y_n. En el caso de esta programación se ha coloreado x_{n+1} de rojo e y_n de azul. Aquí se utiliza, para ser más claros y concisos, un objeto **expr** que permite implementar una fórmula

matemática compleja evitando así utilizar diferentes objetos para cada operación. Así la fórmula $x_{n+1} = 1 - a\ x_n^2 + y_n$ se encuentra implementada en su totalidad utilizando el objeto **expr 1.- (in3 * in1 * in1) + in2** y, como se puede observar, a ese objeto ingresan dos recursiones: **history x 01** e **history y 0.1.** También puede verse que $y_n = b\ x_n$ se encuentra, de igual manera, implementado dentro de un **expr**.

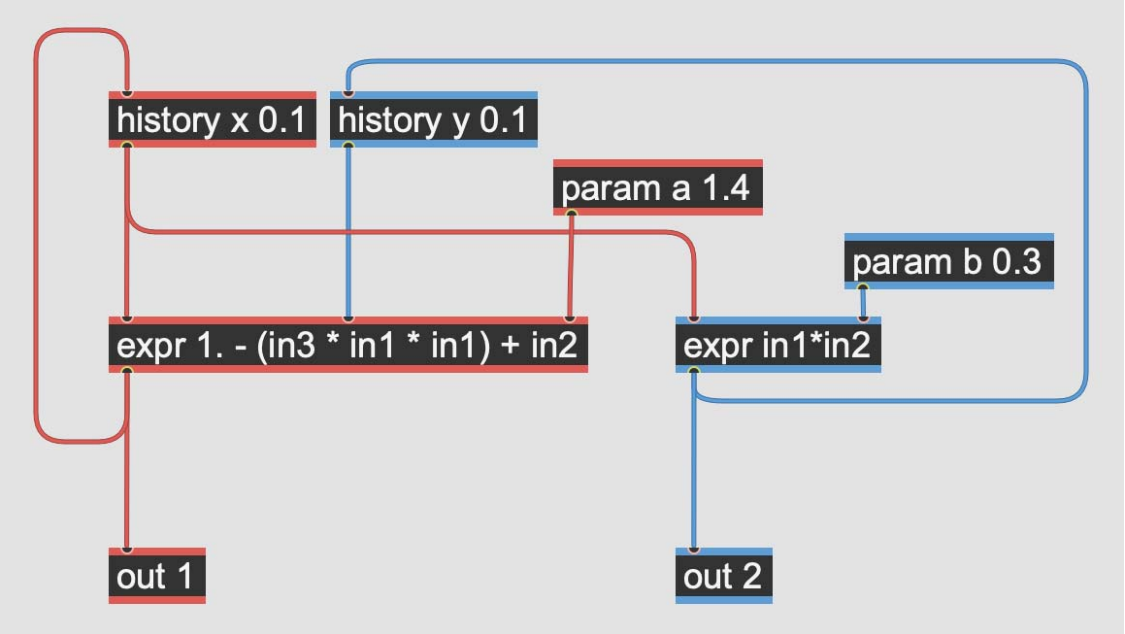


Figura 129: implementación de caos Henon

En esta implementación el valor de *b* permanece constante en 0.3 mientras que *a* varía desde 0 hasta 1.4 (el parámetro se encuentra normalizado a 0. - 1.) La razón para no variar *b* es que, cuando este supera el valor de 0.3, la función se vuelve volátil y las órbitas pierden su delimitación. A continuación los últimos diez valores de cien iteraciones de esta función cuando *a* = 1.4 y *b* = 0.3:

1.26713

-1.27711

-0.903262

-0.525366

0.342608

0.678058

0.459116

0.908315

-0.017317

1.27207

-1.27064

Y los mismos diez valores cuando *a* = 1.4 y *b* = 0.32:

$$-1.387563492412982 \times 10^{602}$$
$$-2.695465423668239 \times 10^{1204}$$
$$-1.017174739026739 \times 10^{2409}$$
$$-1.448502229599762 \times 10^{4818}$$
$$-2.937422192817675 \times 10^{9636}$$
$$-1.207982879440092 \times 10^{19273}$$
$$-2.042911691828527 \times 10^{38546}$$
$$-5.842883452853572 \times 10^{77092}$$
$$-4.779500186108211 \times 10^{154185}$$
$$-3.198107084061179 \times 10^{308371}$$
$$-1.431904448957122 \times 10^{616743}$$

Como se puede apreciar, los valores dan pasos agigantados con cada iteración. Además, variar *b* entre los valores 0 y 0.3 no aporta gran variedad al tipo de perfil generado. En la

siguiente figura se muestra el gráfico de bifurcaciones de esta interpolación (con *a = 1.4).*

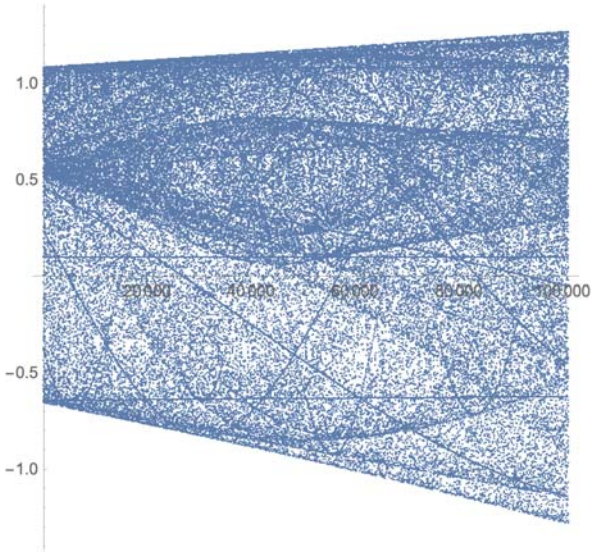


Figura 130: bifurcaciones de la órbita x cuando *a* = 1.4 y *b* varía desde 0.1 a 0.3

En cambio el gráfico de bifurcaciones que se observa al mantener *b* = 0.3 y variar *a* desde 0 a 1.4 es el siguiente:

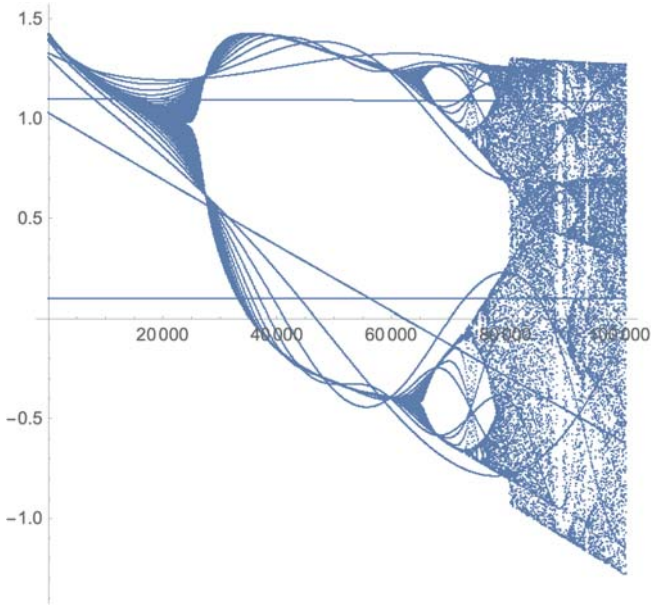
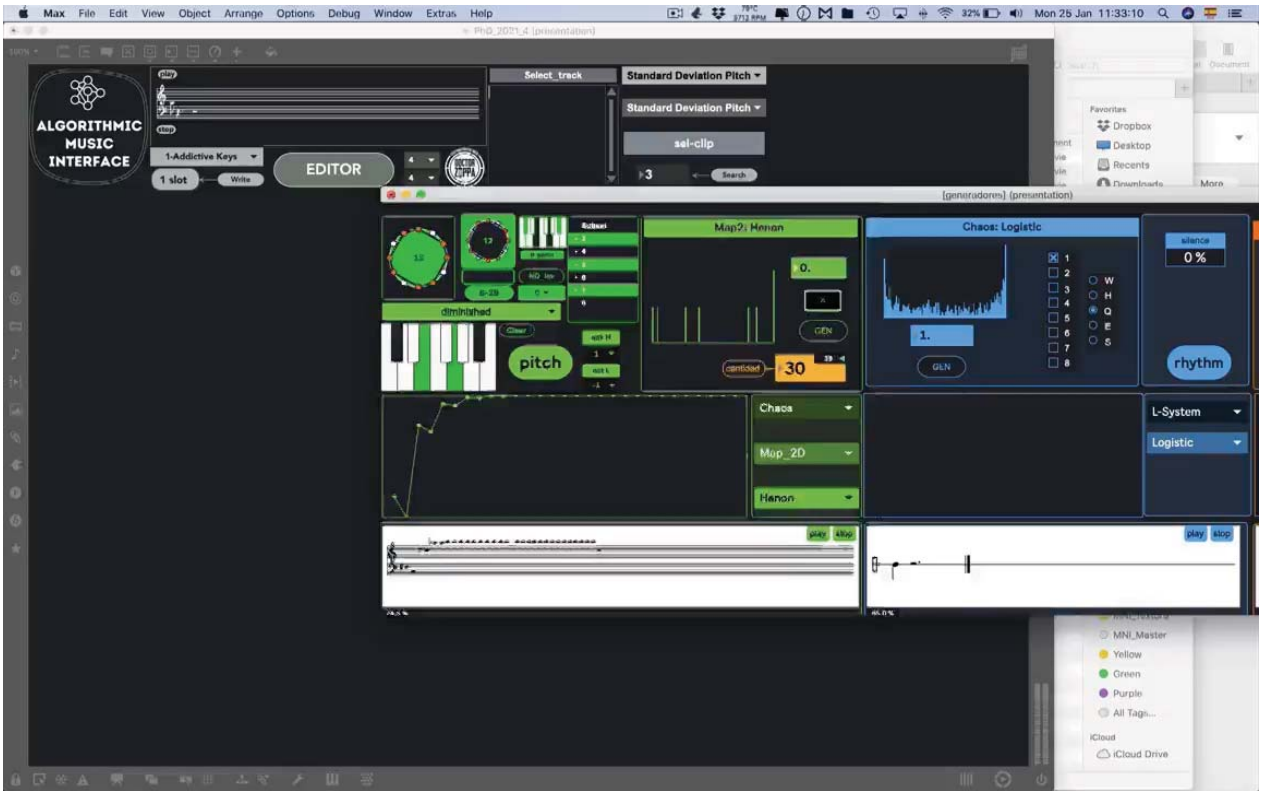


Figura 131: bifurcaciones de la órbita x cuando *b* = 0.3 y *a* varía desde 0. hasta 1.4

Como es posible observar en el video 19, hasta alrededor a = 0.75, la órbita se bifurca como se ve en la figura 131. A partir de allí su estado se vuelve caótico.



Video 19: diferentes valores de *a* para caos Henon

También se aprecia que en la interfaz hay un botón marcado con una X que al presionarlo hace que el perfil generado se produzca a partir de una u otra órbita (X / Y).

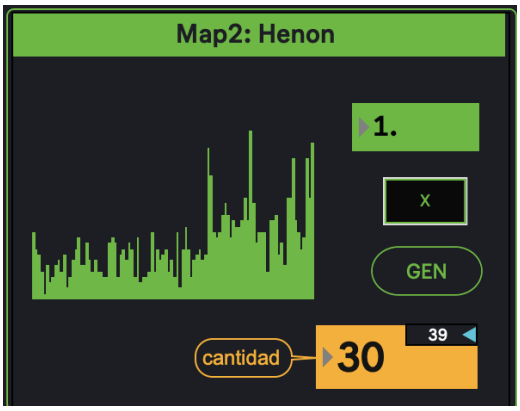


Figura 132: interfaz de caos henon

Henon phase

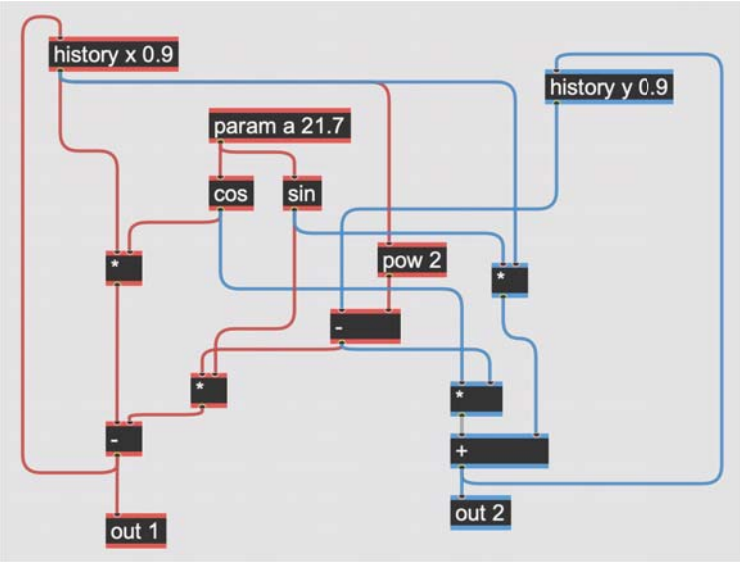


Figura 133: implementación de caos Henon phase

Como es posible constatar en la figura 133, este sistema posee una sola variable (*a*). Puede observarse sin embargo en el video 5 del capítulo 2, que las órbitas *x* / *y* de este sistema caótico evolucionan de manera muy similar a medida que *a* se modifica. También se ve en el mismo video lo que sucede al variar el valor inicial de *x*. Si bien el gráfico de bifurcaciones es más claro, los valores generados varían en un ciclo demasiado repetitivo y por lo tanto poco interesante.

En el video 20 se ve que, cuando *a* = 1, o posee un valor cercano a 1, se produce un perfil autosimilar con ciertas variaciones en la evolución de la forma (esto es válido para ambas órbitas), mientras que con valores inferiores (por ejemplo *a* = 0.46) se produce una especie de onda triangular modulada en su amplitud por un LFO con forma de onda tipo sinusoide.

Video 20: generación de alturas utilizando Henon phase

Clifford

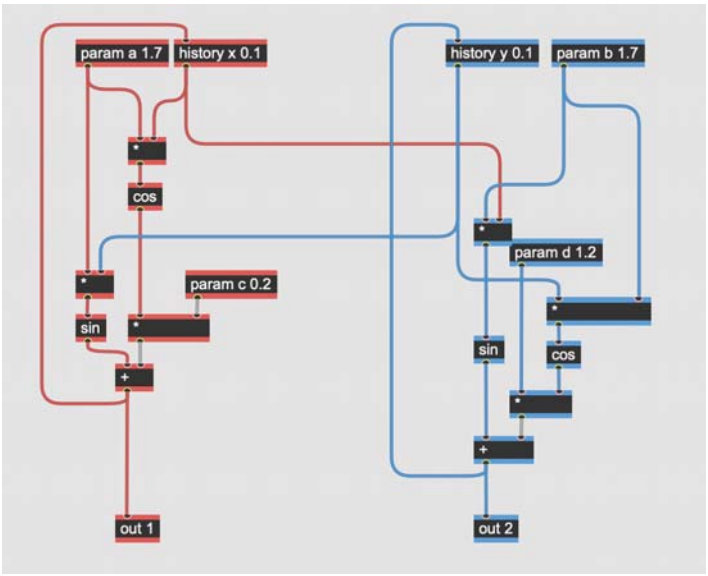
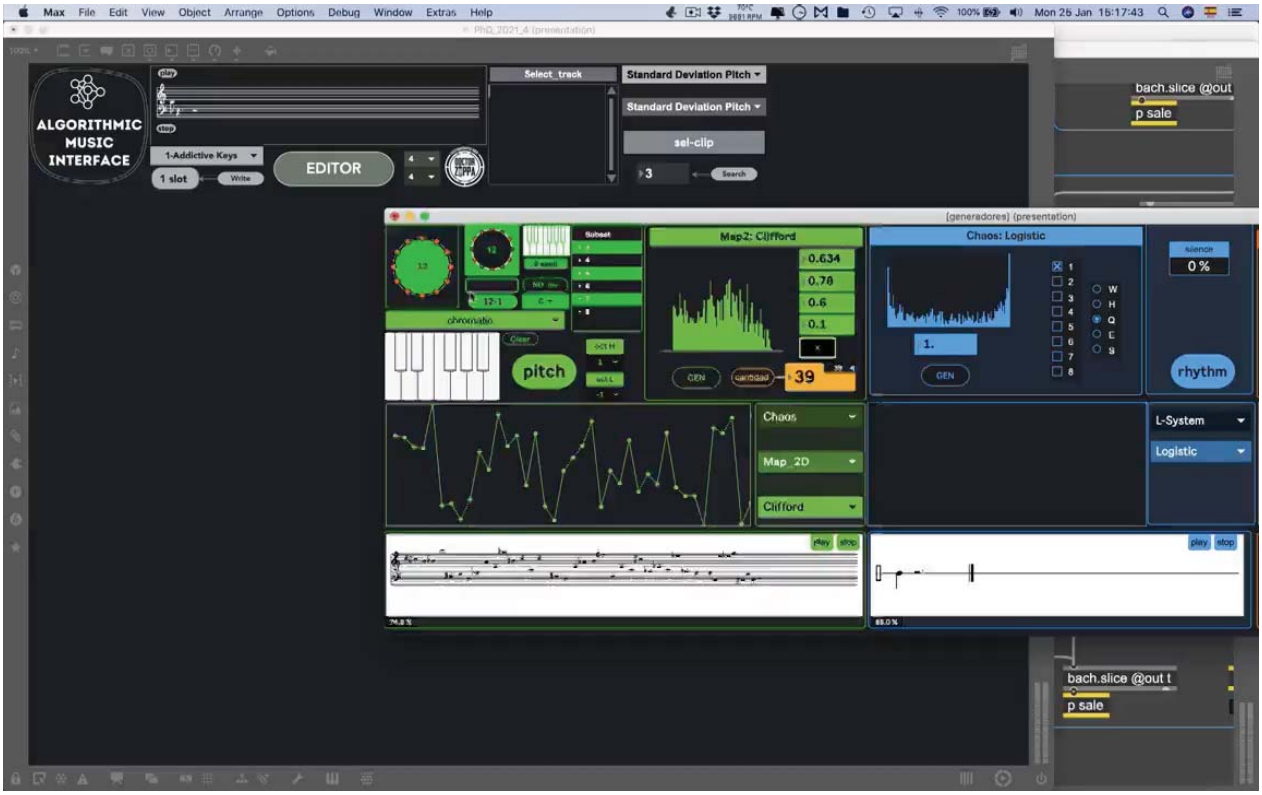


Figura 134: implementación de caos clifford

En el capítulo 2 se señaló que es posible variar aquí cuatro parámetros diferentes: *a, b, c* y *d.* En la figura 134 se aprecia su implementación. Es difícil en este caso prever el resultado que provocará la interacción de la variación de estos parámetros, pero es posible observar en el video 21 que con este generador se pueden obtener patrones cíclicos y caóticos autosimilares.



Video 21: variaciones del generador de Clifford

Jong

Este algoritmo (figura 135) tiene la característica de generar variaciones cuasi cíclicas. Como se ve en el video 22, al modificar sus tres parámetros se obtiene, en casi todos los casos y para ambas órbitas, perfiles que se repiten con mayor o menor variación sin perder la identidad de cada ciclo.

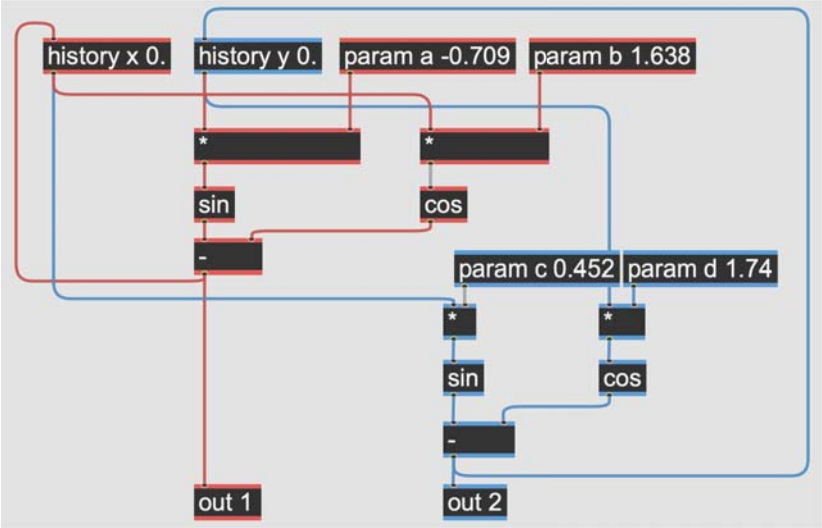


Figura 135: implementación de caos Jong

Video 22: variaciones del generador jong

Gingerbread

Este algoritmo es un poco menos interesante que los anteriores ya que genera mayormente patrones cíclicos con poca variación entre ellos lo que puede resultar un poco monótono. Aquí no se tiene un parámetro que variar por lo que se implementó un sistema para que sea posible variar el valor inicial de *x*.

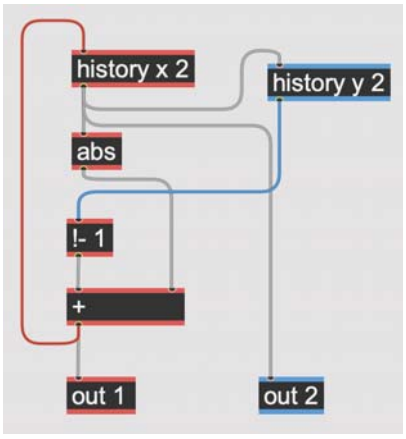
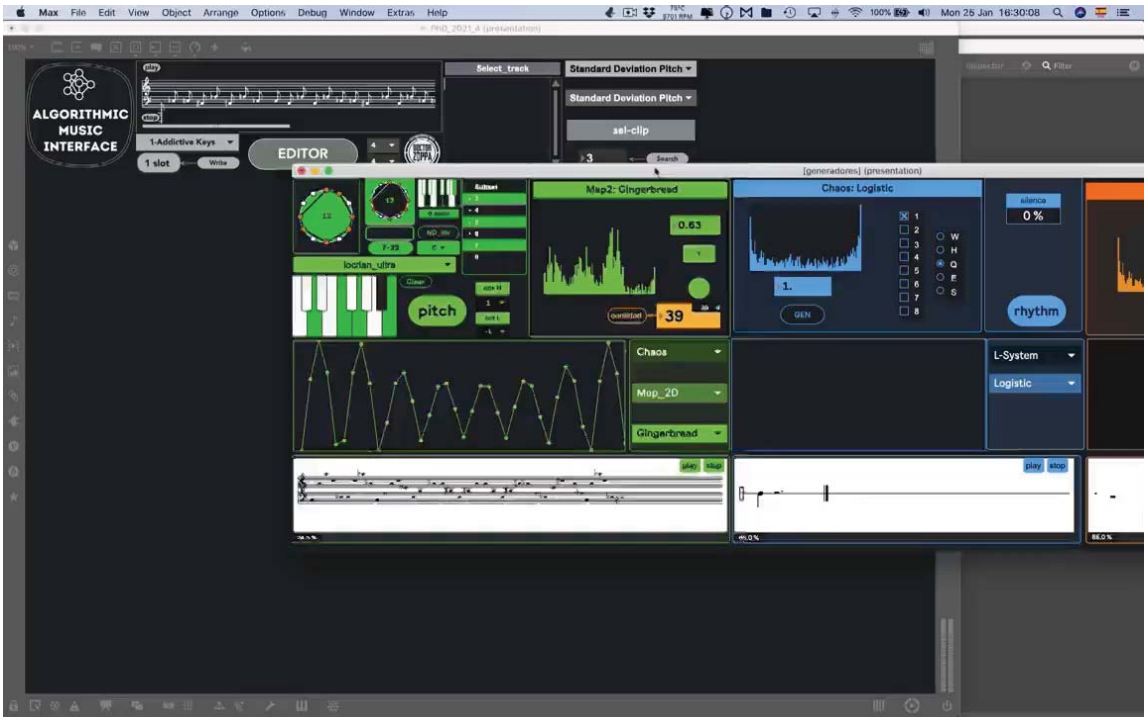


Figura 136: implementación de caos Gingerbread



Video 23: alturas generadas con un algoritmo Gingerbread

De todas maneras se incluyó dentro de los generadores de dos dimensiones porque las órbitas que se obtienen, si bien no cubren un amplio rango de posibilidades, pueden resultar útiles para generar cierto tipo de perfiles de tipo minimalista.

Burning ship

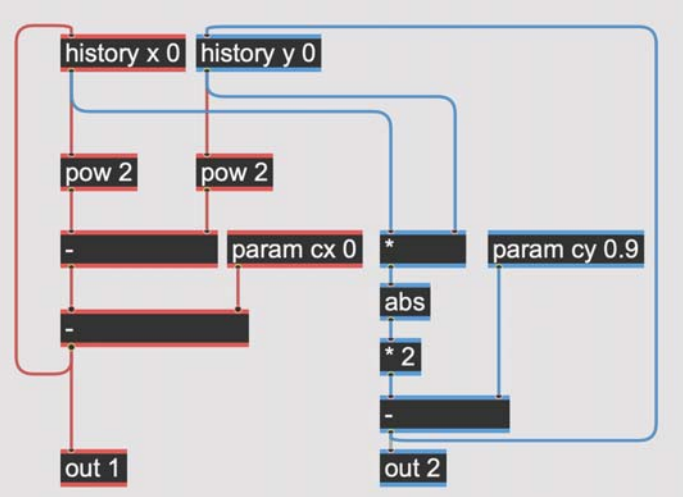
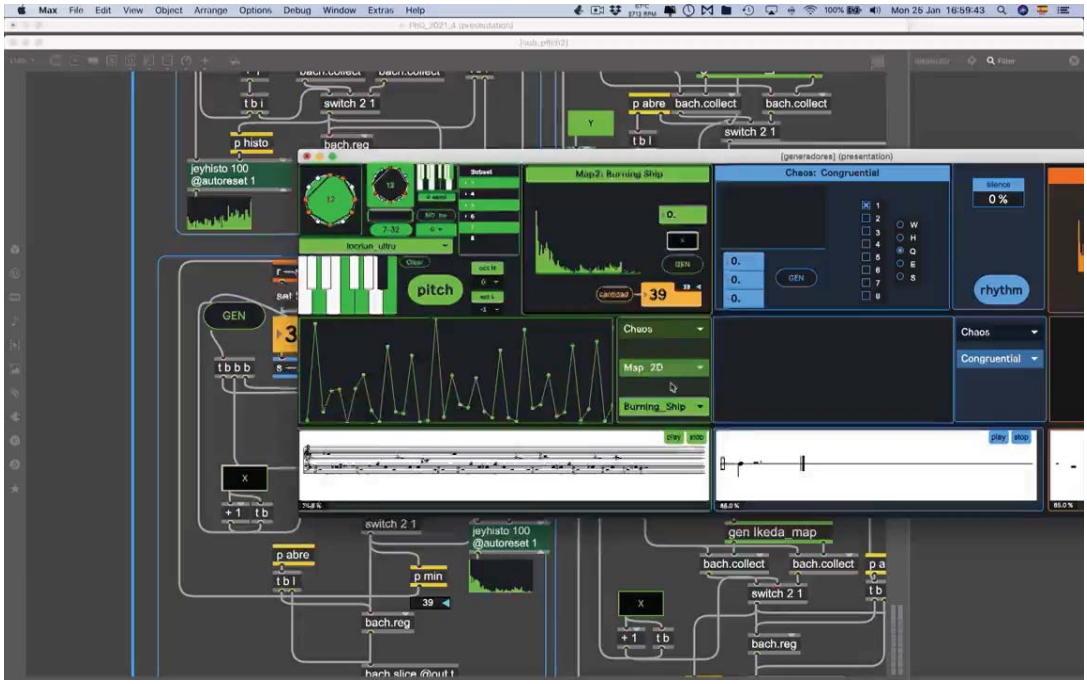


Figura 137: implementación de caos burning ship



Video 24: generador burning ship

Este algoritmo, al igual que en el de gingerbread, genera patrones cíclicos con un nivel alto de auto-similitud.

Ikeda

Como puede observarse en la ecuación 18, este algoritmo es uno de los más complejos de toda esta serie. Por lo tanto en este caso se ha utilizado, como ya se había hecho con *henon*, el objeto **expr** para implementar expresiones matemáticas más complejas. Además aquí se hace uso de los objetos **send** y **receive** (que se pueden utilizar con su abreviatura **s** y **r**). Estos permiten enviar informacion de manera "inalámbrica", es decir que un **send** con una palabra especifica como argumento se comunicará con el **receive** que posee, como argumento, la misma palabra. En este caso, por ejemplo, el **s tox** se comunica con **r tox**. De esta manera se evita el uso excesivo de cables haciendo el patcher mas legible como se observa en la siguiente figura.

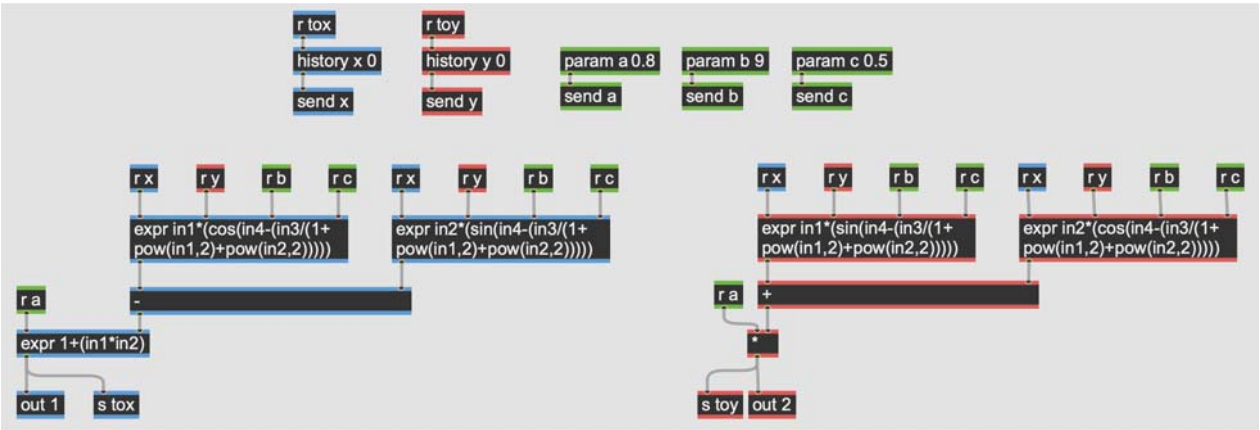


Figura 138: implementación del algoritmo encargado de generar órbitas de ikeda

Es interesante destacar que, cuando el parámetro *a* se encuentra lejos del valor 1, los perfiles generados a partir de la manipulación de los demás parámetros pueden entenderse como caóticos y autosimilares. Si embargo, cuando *a* se aproxima a 1, se dibuja un perfil con forma de sinusoide que puede alterarse al modificar los parámetros *b* y *c.* En el video 25 se puede apreciar lo antedicho.

Video 25: generación de alturas a través de Ikeda

Lorenz

Como ya se mencionó en el capitulo 2, este mapa es de tres dimensiones y en su fórmula entran en juego las ecuaciones diferenciales. También se explicó que el método de Euler para su resolución es suficientemente preciso para los fines de esta implementación.

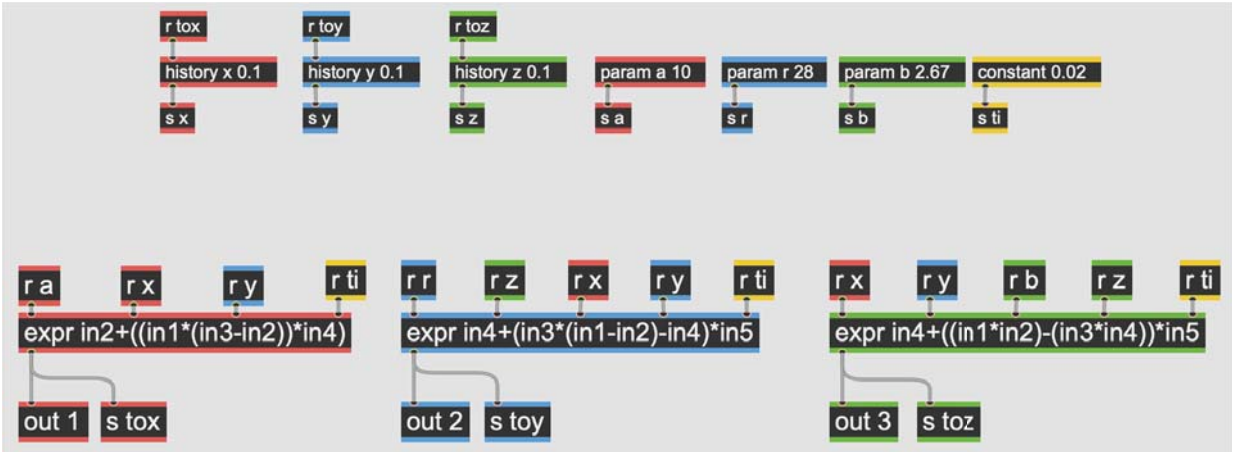


Figura 139: implementación de caos lorenz

Como puede observarse, aquí también se hace un uso extensivo de los objetos **expr**, **send** y **receive**.

Video 26: alturas generadas con lorenz

Si bien este algoritmo cuenta con tres variables (σ, ρ, β), se agregó también la posibilidad de modificar el valor de delta, es decir, la resolución del método de Euler. Así es posible variar desde un perfil más suave a uno más cambiante. También se puede aquí utilizar cualquiera de las tres órbitas generadas. Delta es, en esta implementación, el parámetro que se encuentra en la parte superior de la interfaz de este algoritmo.

Aleatorios

La implementación de estos algoritmos que se presentan a continuación no requiere de funciones recursivas, por lo tanto su programación no se realiza dentro del objeto **gen** como en el caso de los sistemas caóticos. Se muestran a continuación los seis tipos de sistemas aleatorios que se han utilizado en el presente trabajo para generar alturas.

Random walk

Como se observa en la figura 140 en el caso de random walk no hay parámetros para modificar ya que los valores de este sistema se obtienen, como se vio en el capítulo 2, generando aleatoriamente un paso hacia adelante o un paso hacia atrás (1, -1).

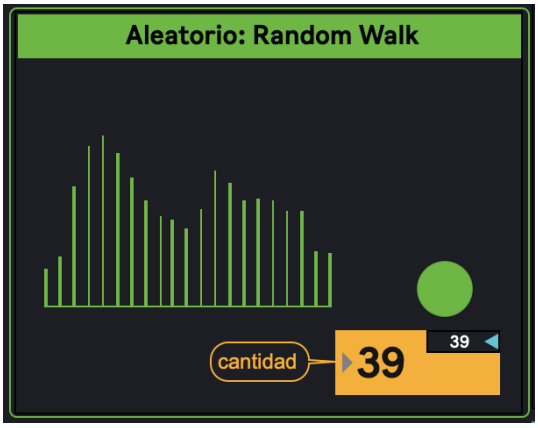


Figura 140: interfaz para random walk

Como se constata en la figura 141, la programación es extremadamente simple. Un objeto **random** genera dos valores posibles (0 - 1) de manera aleatoria. Estos valores atraviesan un objeto condicional **if $i1 == 1 then 1 else 0** que, cuando recibe un 1 este sale sin cambios pero si recibe un 0 envía un -1. Luego estos valores se acumulan dentro del objeto **accum** para poder así obtener el perfil buscado. Por ejemplo, si desde el condicional se reciben los siguientes valores: -1, 1, 1, 1, 1, -1, -1, 1… la acumulación será -1, 0, 1, 2, 3, 2, 1, 2.

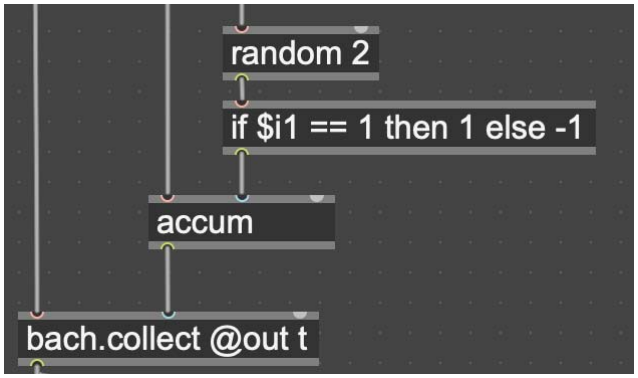


Figura 141: implementación de random walk en *Max*

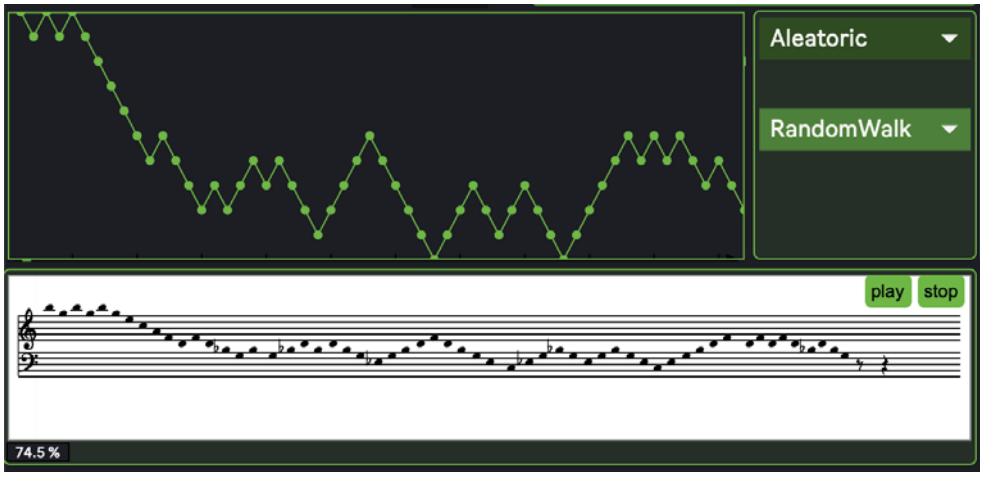


Figura 142: ejemplo de random walk

Lineal

En la figura 143 izquierda se observa cómo se hace uso aquí de dos objetos **random 1001** (que generarán valores aleatorios entre 0 y 1000). Dichos generadores pasan a través del objeto condicional **if $i1 < $i2 then $i1 else $i2** y de esta manera, si el valor que ingresa por la primera entrada es menor al que ingresa por la segunda se escoge el primero, sino el segundo. En el ejemplo de la derecha se implementa de manera inversa. Se puede observar que los histogramas obtenidos se corresponden con los esperados[91].

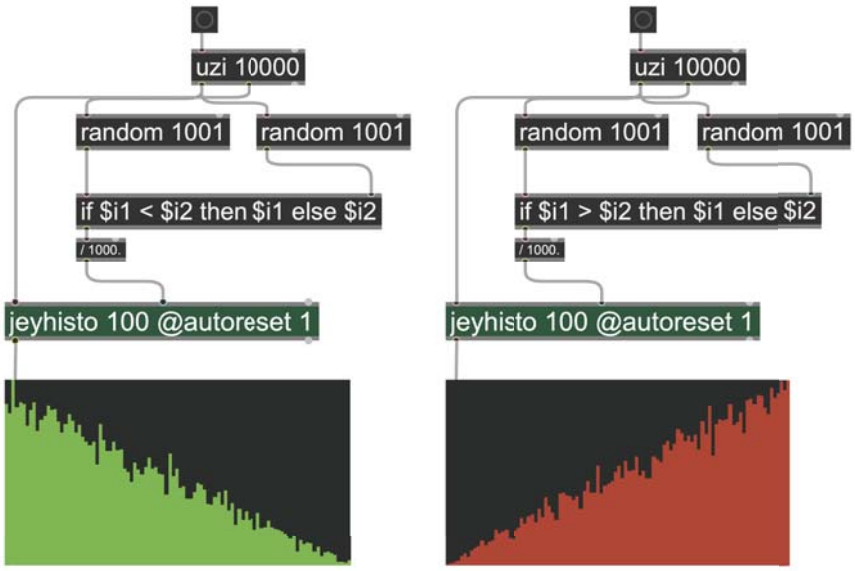
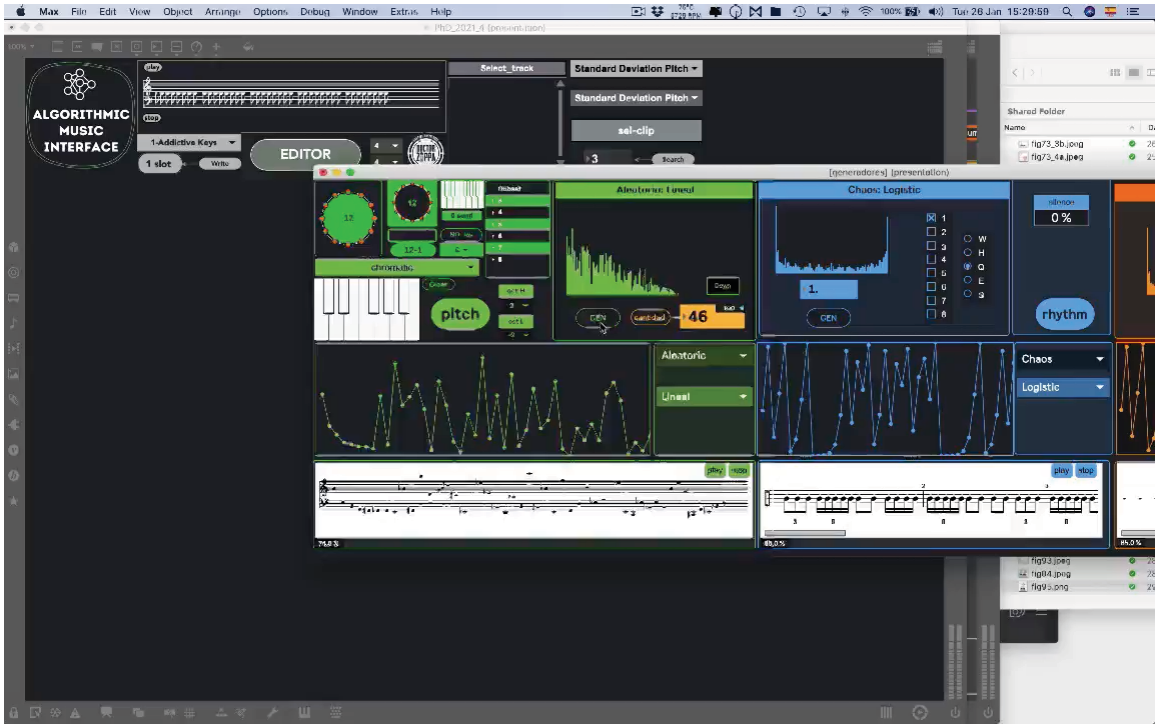


Figura 143: random lineal

El resultado de estos dos generadores es la obtención perfiles aleatorios (ya no autosimilares como en varios de los casos vistos hasta aquí) pero dicho perfil tenderá a generar notas dentro del rango grave (en el caso del generador de la izquierda) y agudas en el caso de la derecha.

91 Para obtener dichos histogramas utilizamos el objeto jeyhisto desarrollado por Jeyong Jung, Johan van Kreij, Peter Pabon y Sohrab Motabar. https://sourceforge.net/projects/jey-toolkit/

En el ejemplo del video 27 se está utilizando una escala cromática y un rango amplio (de cuatro octavas) para que sea más evidente el efecto que produce un generador o el otro.



Video 27: ejemplo de sistema algorítmico lineal

Triangular

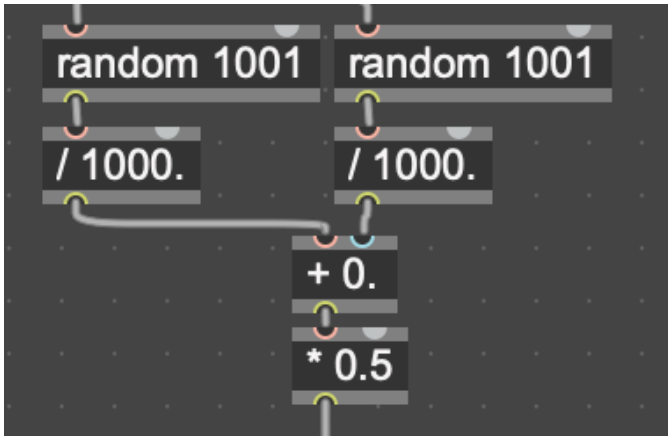


Figura 144: aleatoriedad triangular

Este generador utiliza los valores promedio producto de la suma de dos objetos aleatorios flotantes (0. - 1.). Es por ello que la implementación es muy sencilla y producirá, a diferencia de la aleatoriedad lineal, valores que se encuentran mayormente en la parte media del ámbito escogido.

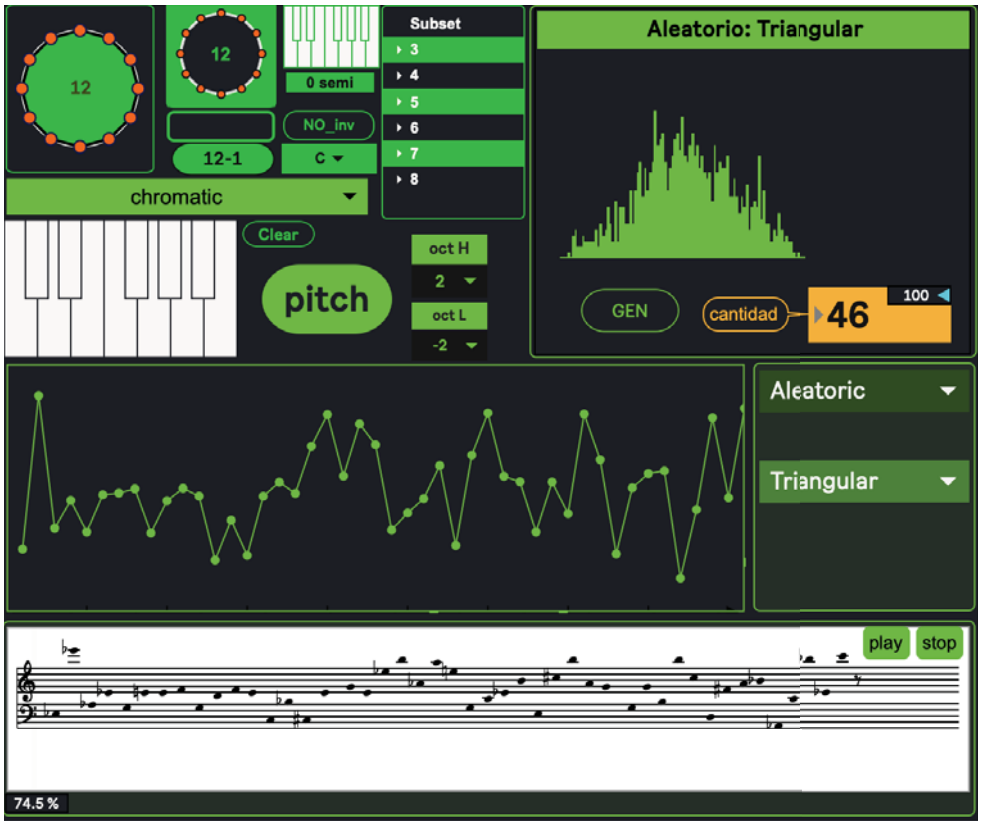
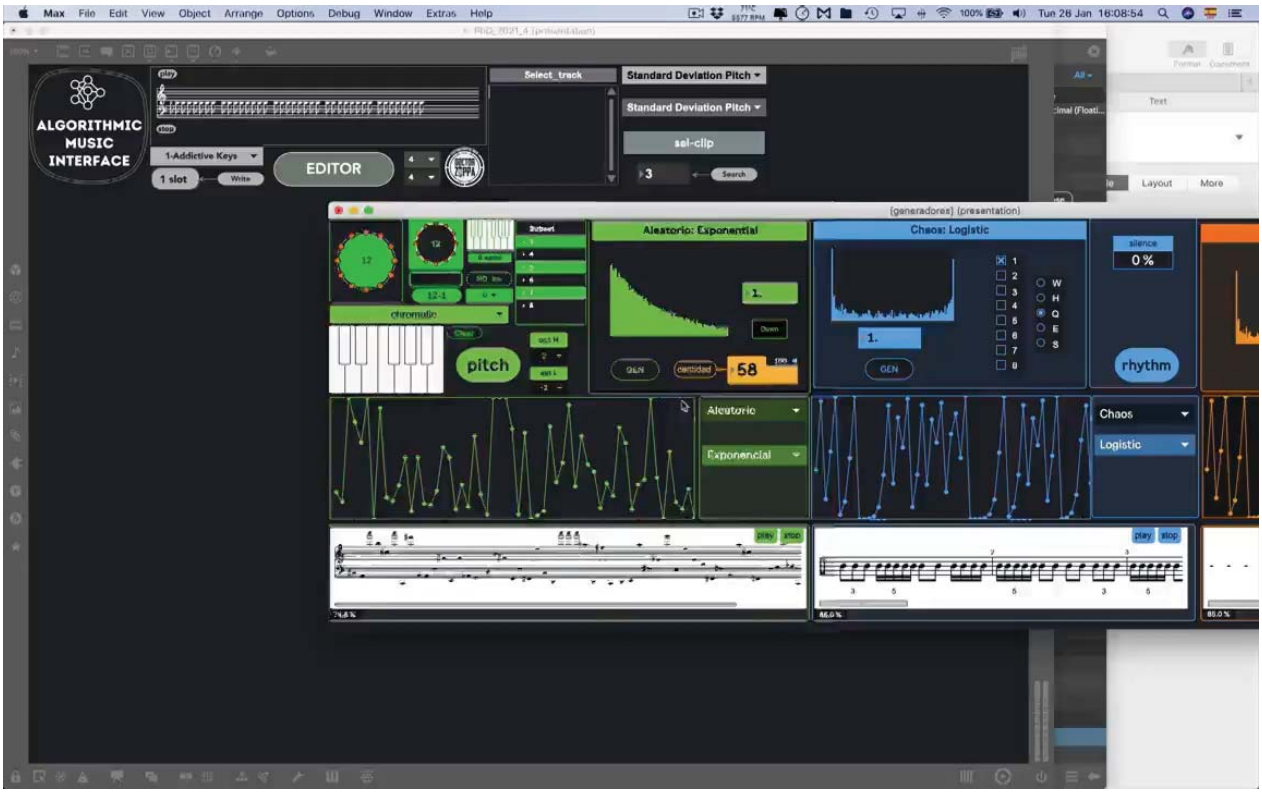


Figura 145: interfaz de aleatoriedad triangular

Exponencial

Este generador produce un resultado similar al lineal ascendente o descendente, con la particularidad de que es posible especificar el nivel de curvatura que tendrá dicha pendiente. De esta manera puede acentuarse aún más las probabilidades de que los valores generados se encuentren en mayor medida en el rango agudo o en el grave.



Video 28: ejemplo de aleatoriedad exponencial

Bilateral

Este generador es al triangular lo que el exponencial es al lineal, es decir, que genera valores medios pero se puede ajustar la curva de las dos pendientes del triángulo.

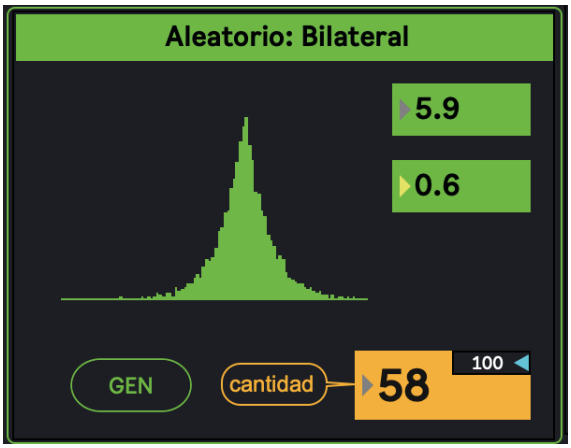


Figura 146: interfaz de random bilateral

Desde el parámetro superior (figura 146) es posible modificar dicha curva, mientras que desde el parámetro inferior puede moverse el centro del triángulo para así definir el registro donde se generarán los valores.

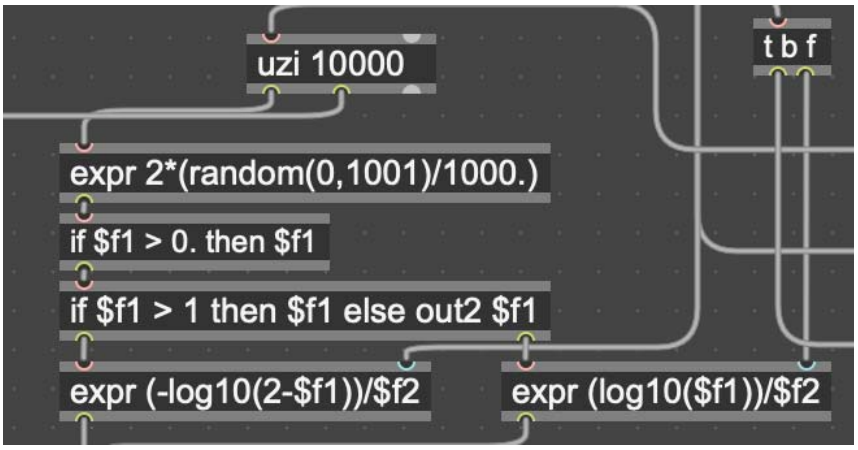
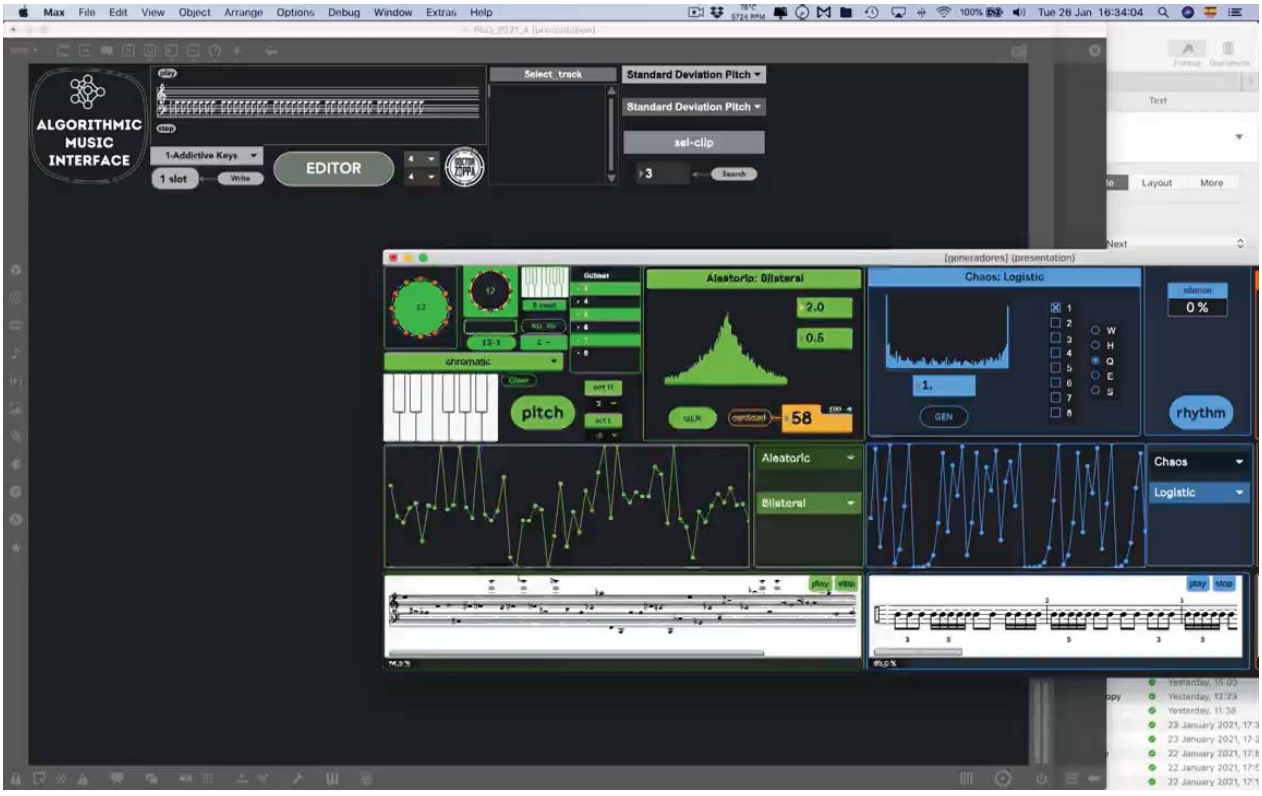


Figura 147: programación de aleatoriedad bilateral



Video 29: ejemplo de random bilateral

En la figura 147 se puede ver que se utilizó un objeto **expr 2*(random(0,1001)/1000.)** para generar los valores aleatorios entre 0 y 2, luego el condicional **if $f1 > 0. then $f1** evita que el valor 0 ingrese al algoritmo. Esto se debe a que $Log(0) = -\infty$. Luego los valores generados atraviesan otro condicional **if $f1 > 1 then $f1 else out2 $f1** que envía los valores mayores a 1 por la salida izquierda y el resto por la derecha. Así los valores que se encuentran en el rango $0 < x < 1$ pasan a través del objeto **expr (-log10(2-$f1))/$f2** y los que están dentro de $1 \leq x < 2$ pasan a través del objeto **expr (log10($f1))/$f2**.

Gauss

Este generador arroja valores que poseen una distribución de Gauss. El resultado es similar a la distribución triangular con la diferencia de que sus pendientes forman una función cóncava. Como se ha visto en el capítulo 2, es posible generar esta función a partir de la siguiente fórmula:

$$\sigma * (\sqrt{-2\ Log(random)} * sin(2\ \pi\ random) + \mu$$

Ecuación 30

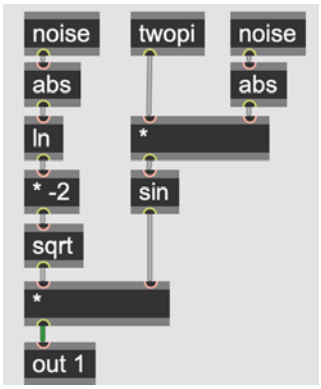


Figura 148: Gauss implementado en *Max*

Su implementación en *Max* es la que se muestra en la figura 148.

Genético

La programación de este algoritmo en la herramienta *AMI* es quizá la que tomó más trabajo de diseño. La explicación de como se realizó brindará al lector los pormenores de cómo se llevo a cabo dicha implementación.

Generación

Como se mostró en el capítulo 2, el primer paso de este algoritmo consiste en la generación aleatoria de cromosomas (ver figura 75). En este caso, estos cromosomas serán una combinación aleatoria de las notas escogidas a partir del módulo para seleccionar y modificar alturas explicado en el punto 3.2.1. El tamaño del cromosoma dependerá del valor estipulado por el usuario, como así también el ámbito que abarcará.

A partir de lo antedicho se desprende que lo primero que se debe hacer al utilizar este algoritmo es especificar el grupo de alturas que se utilizarán y su ámbito de octavas. Una vez realizado esto, se debe escoger el tamaño del cromosoma desde el valor indicado con un triángulo amarillo como se observa en la figura 149 y que en este caso es de 8. Luego es posible especificar la cantidad de notas que se desean generar[92].

92 Recordar que el valor del entero que se encuentra sobre el número en amarillo (en negro con un triángulo azul) indica la cantidad máxima de notas que puede generar una algoritmo determinado.

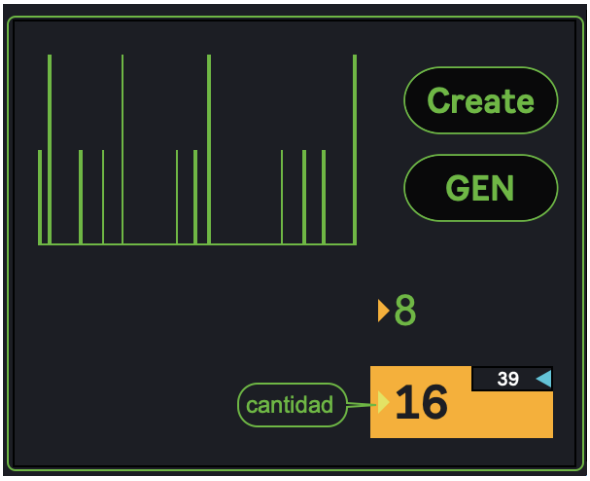


Figura 149: interfaz para el algoritmo genético

La generación aleatoria de cromosomas se produce en la parte del patch que se observa a continuación:

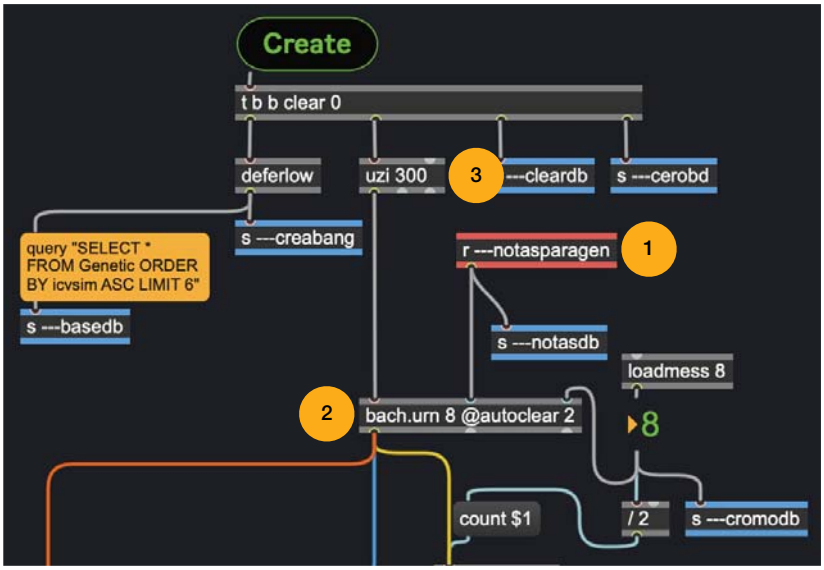


Figura 150: implementación de la generación de cromosomas aleatorios (detalle)

Lo que se observa en la figura 150 es un detalle de un sector de la programación correspondiente a la implementación del algoritmo genético. En la siguiente figura se ve la programación completa y, dentro del recuadro rojo, el sector que se muestra en la figura 150.

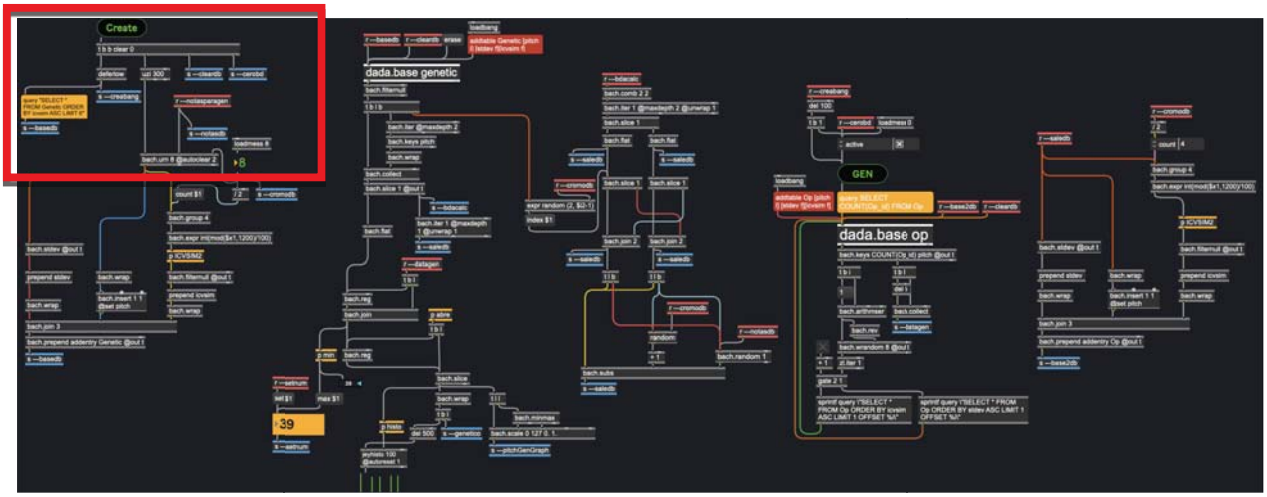


Figura 151: implementación de la generación de cromosomas aleatorios

Cuando se escoge un grupo de alturas desde la interfaz correspondiente, por ejemplo una escala de tonos enteros [0 2 4 6 8 10] con un ámbito de dos octavas, las notas del conjunto serán (si se comienza en Do):

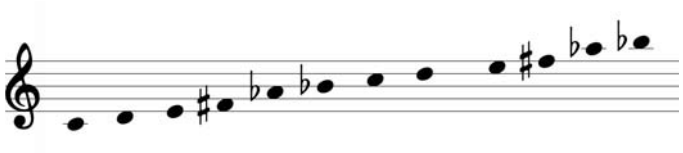


Figura 152: escala de tonos enteros que abarca dos octavas

Estas alturas, en valores MIDI, llegan al sector del patch que se ve en la figura 150 a través del objeto **r —-notasparagen** (1). Esta lista [60 62 64 66 68 70 72 74 76 78 80 82], se acumula dentro del objeto **bach.urn 8** (2) que, cuando recibe un estímulo[93] en su entrada izquierda, escogerá de entre estos, ocho valores al azar sin repeticiones. Cuando se acciona el botón “Create”, el objeto **uzi 300** envía trescientos de estos estímulos y así se generarán trescientos cromosomas.

93 Denominados bangs en *Max*

Cálculo de aptitud

A continuación se debe calcular la aptitud de cada uno. En este caso se hará a partir de la desviación estándar y la relación IcVSIM interna de cada lista. La desviación estándar de una lista de valores es la medida de dispersión que posee dicha lista con respecto a la media y se calcula a partir de la raíz cuadrada de la varianza (Salazar & Castillo, 2018):

$$\sqrt{\frac{\sum \left(x_i - \mu\right)^2}{N}}$$

Ecuación 31

donde x_i es el valor $i$ésimo de la variable que se está calculando, y μ es la media de la población:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} \mu_i$$

Ecuación 32

A partir de este valor se pueden identificar cadenas de notas que posean perfiles más dispersos que otros. Por ejemplo, en la figura 153 se aprecian una serie de notas que tienen como valor medio el Do (línea roja). Como puede observarse, el grupo de la derecha arroja una desviación estándar mucho mayor que el de la izquierda.

Respecto a la relación IcVSIM, esta fue propuesta por Eric J. Isaacson como una manera de obtener el grado de similitud que poseen dos Pitch Class diferentes sin importar su cardinalidad (Isaacson, 1990). Este autor analiza propuestas de diferentes investigadores

(Allen Forte, Charles Lord, Robert Morris, John Rahn, David Lewin y Richard Teitelbaum) y encuentra que ninguna de ella aporta las condiciones necesarias para ofrecer una comparación satisfactoria. Como lo señalan Pablo Di Liscia y Pablo Cetta (Cetta & Di Liscia, 2010) estas condiciones son: proveer un valor distintivo para cada par de conjuntos de la misma clase a comparar, ser utilizable para clases de conjuntos de cualquier cardinalidad y proveer un rango amplio de valores discretos. La propuesta de Isaacson, entonces, satisface estas condiciones.

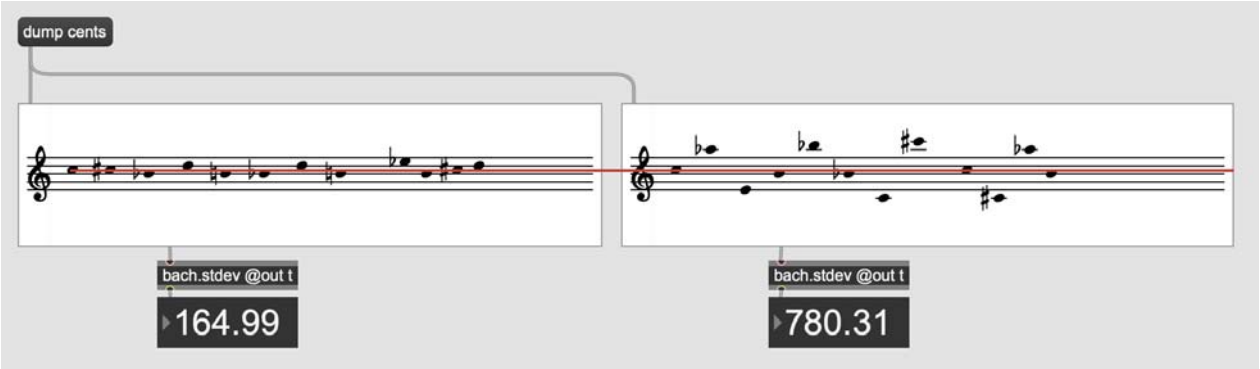


Figura 153: ejemplo de desviación estándar

Para calcular el IcVSIM de dos conjuntos de notas se debe primero calcular el vector de intervalos de cada conjunto. Este vector indica cuántos intervalos de cada tipo se pueden encontrar en un PCs de cualquier cardinalidad. Los intervalos posibles en el sistema temperado son la segunda menor, segunda mayor, tercera menor, tercera mayor, cuarta justa y cuarta aumentada. Los demás serán inversiones de estos (por ejemplo un intervalo de quinta, invertido se convierte en uno de cuarta). Ver la siguiente figura.

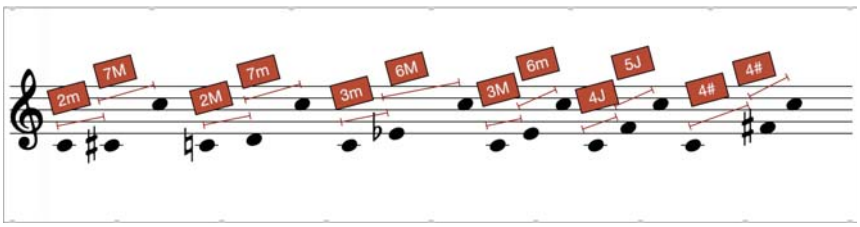


Figura 154: lista de intervalos

Por lo tanto, si se toman un grupo de notas, por ejemplo, [Re Mi Fa# La] los intervalos contenidos dentro de este conjunto serán:

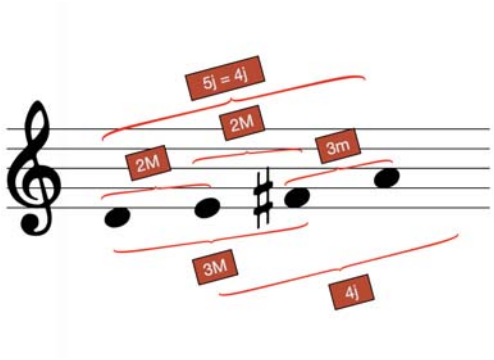


Figura 155: todos los intervalos contenidos en [Re Mi Fa# La]

Como se puede ver en la figura 155, en dicho grupo de notas no hay ningún intervalo de segunda menor, hay dos de segunda mayor, uno de tercera menor, uno de tercera mayor, dos de cuarta justa y ninguno de cuarta aumentada. Por lo tanto su vector de intervalos será [0 2 1 1 2 0]. Si ahora se toma otro grupo de notas de diferente cardinalidad, por ejemplo [Do Mi Fa Fa# Sol# Si], su vector será [3 2 2 3 3 2]. A continuación se debe calcular la diferencia entre ambos vectores (IdV) que será:

3 2 2 3 3 2
-
0 2 1 1 2 0
_____
3 0 1 2 1 2

Ahora se calcula la desviación estándar de este vector, es decir:

$$\mathrm{IcVSIM} = \sqrt{\frac{\sum \left(\mathrm{IdV}_i - \overline{\mathrm{IdV}} \right)^2}{6}}$$

Ecuación 33

Donde IdV_i es el $i$ésimo término de IdV (en este caso 3, 0, 1, 2, 1, 2) y $\overline{\mathrm{IdV}}$ es el promedio de este vector (es decir 1.5 en este caso). Entonces si se realiza el cálculo:

$$\sqrt{\frac{\left((3-1.5)^2+(0-1.5)^2+(1-1.5)^2+(2-1.5)^2+(1-1.5)^2+(2-1.5)^2\right)}{6}}$$

El resultado será 0.957427 que es el valor de IcVSIM que resulta de comparar la similitud entre [Re Mi Fa# La] y [Do Mi Fa Fa# Sol# Si].

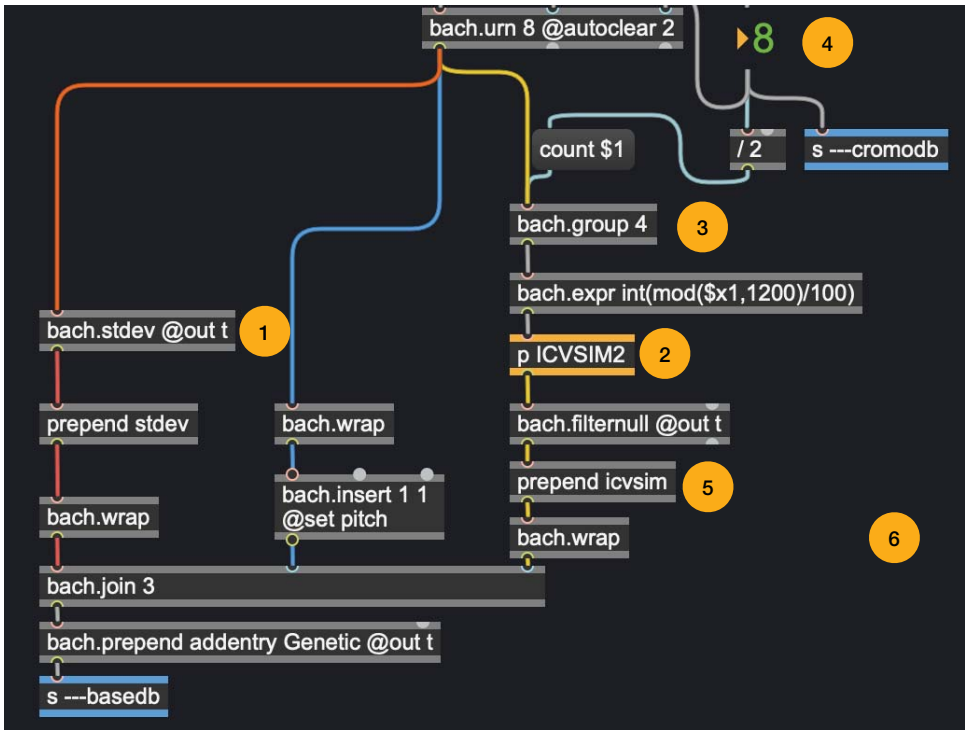


Figura 156: implementación del cálculo de aptitud

En la figura superior se observa la programación que se ha desarrollado para implementar el sistema de selección de cromosomas a partir de sus aptitudes, que, como ya se mencionó, se calculan a partir de su desviación estándar o de su relación IcVSIM interna. En el primer caso se está utilizando un objeto **bach.stdev** (1) que calcula dicho valor de

dispersión. Este objeto implementa simplemente la ecuación 31. Para el coeficiente de IcVSIM el cálculo se produce dentro del **subpatch p ICVSIM2** (2) que será explicado a continuación. Primero cabe aclarar que, si el grupo de notas escogido es menor al tamaño del cromosoma, se producirán repeticiones aunque estas no podrán sucederse más de dos veces. Por ejemplo, si se escogen tres notas [0 1 2] y el tamaño del cromosoma es de ocho, entonces desde **bach.urn 8** podría generarse [0 2 1 0 1 2 2 1] pero nunca [0 1 2 2 2 1 0 0]. Luego los trescientos cromosomas ingresan a **bach.group 4** (3) que separa a cada uno en dos grupos, por ejemplo [Do Re Mib Fa Sol Sol# La Si] quedará [Do Re Mib Fa] [Sol Sol# La Si]. El tamaño de los grupos dependerá del tamaño escogido para el cromosoma, pero siempre habrá dos grupos. Entonces si se escoge un cromosoma de tamaño 10, por ejemplo, ese valor que se indicaría en (4) ingresa al objeto **/ 2** que divide este valor a la mitad y luego atraviesa el mensaje **count $1** que le indica a **bach.group** que ahora debe crear grupos de cinco elementos. Si el tamaño del cromosoma es impar no tiene importancia ya que, aunque los dos grupos serán diferentes, sus vectores siempre tendrán seis elementos.

La librería *Bach* utiliza para las alturas valores MIDI multiplicados por 100 por lo que las notas de la figura 155 serían [6200 6400 6600 6900], por lo tanto se debe reducir estos valores a Pitch Class (0 - 11). Para esto es que se utiliza el objeto **bach.expr int(mod($x1,1200)/100)**, de esta manera dicha lista queda [2 4 6 9].
Cada cromosoma, entonces, separado internamente en dos grupos como ya se señaló, ingresa a **p ICVSIM2** (2) y dentro de este subpatch existe la programación que se ve en la figura 157.

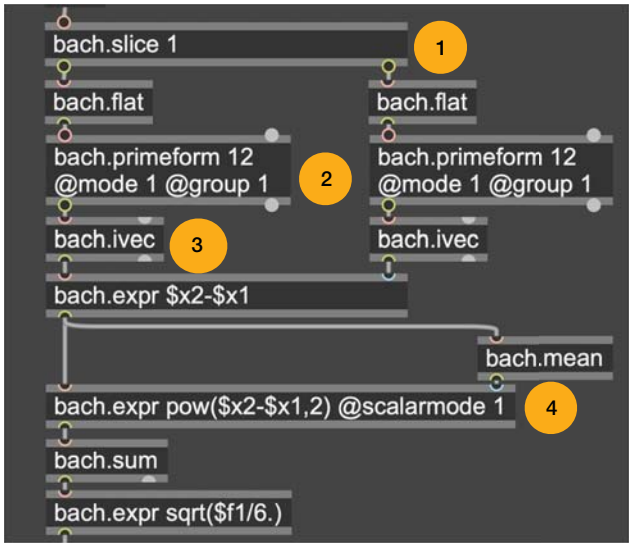


Figura 157: programación dentro de **p ICVSIM2**

Para explicar dicha programación, se puede suponer que uno de los trescientos cromosomas (de 6 valores) generados es [0 2 3 7 2 9]. Éstos se agruparán, como ya se ha visto, de la siguiente forma: [0 2 3] [7 2 9]. Estos dos grupos ingresan a **bach.slice 1** (1) donde se separan saliendo cada uno por una salida diferente. Estas listas, luego de atravesar sendos **bach.flat** para eliminar sus corchetes, ingresan a sendos **bach.primeform** (2) donde se calcula la forma prima de cada grupo. Luego se calcula el vector de intervalos de cada uno en **bach.ivec** (3) y estos son restados en **bach.expr $x2-$x1** para obtener el vector de diferencia de intervalos o IdV, en ese caso quedaría [-1 0 -1 0 2 0]. Por último, para completar la implementación de la ecuación 33 se calcula el promedio de dicho IdV utilizando **bach.mean** y se aplica la fórmula utilizando los objetos que se ven en la figura 158. Estos dos valores de aptitud (desviación estándar e IcVSIM) son almacenados, junto con los alturas correspondientes a dichos valores, en un objeto llamado **dada.base**[94] que utiliza una

94 Los objetos de la librería dada, como así también los de cage has sido desarrollados por los mismos creadores de la librería Bach, es decir Andrea Agostini y Daniele Ghisi. https://www.bachproject.net

base de datos tipo *SQLite* que "es una librería en lenguaje C que implementa un motor de base de datos *SQL* pequeño, rápido, autónomo, de alta confiabilidad y con todas las funciones necesarias. *SQLite* es el motor de base de datos más utilizado del mundo. *SQLite* está integrado en todos los teléfonos móviles y en la mayoría de las computadoras y viene incluido dentro de innumerables otras aplicaciones que la gente usa todos los días"[95].

Figura 158: implementación de la fórmula para obtener el IcVSIM

Este tipo de bases de datos permite realizar consultas en las que se filtra cierta información o se ordena de una manera determinada. Por ejemplo, es posible pedirle a **dada.base** que nos regrese los cromosomas según su índice de IcVSIM de manera creciente. A continuación un ejemplo de cómo se almacenan los datos dentro de **dada.base**:

```
[ Genetic
        [ pitch l ]
        [ stdev f ]
        [ icvsim f ]
]
[
        [
                [ Genetic_id 1 ]
                [ pitch 6500 6600 6800 6400 6100 6000 6300 7000 ]
                [ stdev 315.980616 ]
                [ icvsim 0.816497 ]
        ]
        [
                [ Genetic_id 2 ]
```

95 https://www.sqlite.org

```
        [ pitch 6300 6000 6100 6400 7000 6500 6600 6800 ]
        [ stdev 315.980616 ]
        [ icvsim 0.816497 ]
]
[
        [ Genetic_id 3 ]
        [ pitch 7000 6600 6500 6100 6400 6300 6000 6800 ]
        [ stdev 315.980616 ]
        [ icvsim 0.57735 ]
]
[
        [ Genetic_id 4 ]
        [ pitch 6400 7000 6000 6800 6600 6300 6100 6500 ]
        [ stdev 315.980616 ]
        [ icvsim 1.154701 ]
]
```

Como es posible observar, la base de datos posee una tabla (nombrada en este caso como “Genetic”) y cuatro columnas: “Genetic_id “, “pitch”, “stdev” y “icvsim”. Cada columna almacena diferentes datos. La primera, la posición en la tabla de los datos específicos; la segunda, la lista de alturas generadas; la tercera, la desviación estándar de dicha lista y la cuarta, su IcVSIM. Así si se envía, por ejemplo, el mensaje ***query "SELECT * FROM Genetic ORDER BY icvsim ASC LIMIT 6"*** a **dada.base** este devolverá las primeras seis listas de alturas que posean el índice IcVSIM más pequeño de manera ascendente, es decir, las seis primeras que sean simétricamente más similares. Por ejemplo, una de estas generaciones arrojó los siguientes valores (recordar que son valores MIDI multiplicados por 100)

A = [7000 6600 6500 6000 6100 6800 6300 6400]

B = [6000 6300 6600 6100 6800 6500 6400 7000]

C = [7000 6100 6500 6000 6400 6600 6800 6300]

D = [7000 6100 6500 6400 6600 6300 6000 6800]

E = [6000 6600 6800 6500 6300 7000 6400 6100]

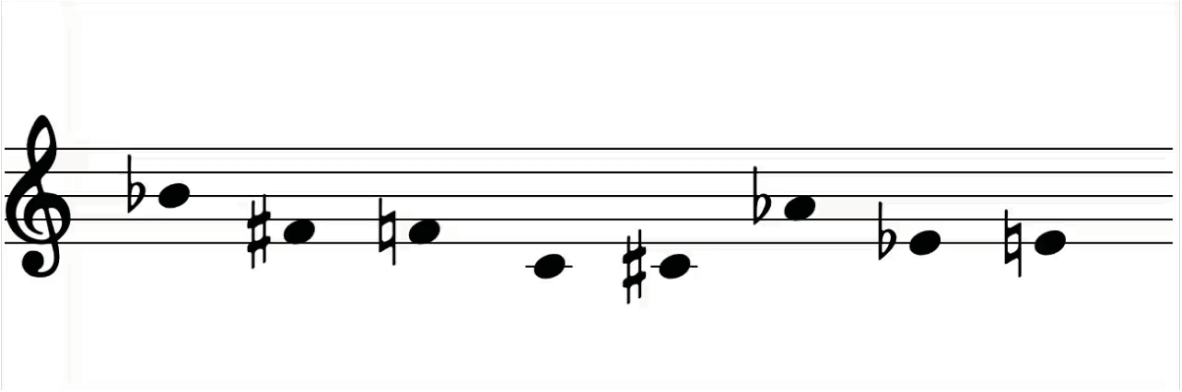
F = [6300 6800 6000 6600 7000 6400 6100 6500]

Si se toma el primer grupo y se analiza el vector de intervalos de sus dos mitades se obtiene:

[7000 6600 6500 6000] = [1 1 0 1 2 1]

[6100 6800 6300 6400] = [1 1 1 1 2 0]

Su IdV es de 0.57735. En el siguiente video se ven estas relaciones de intervalos:



Video 30: relación entre los vectores de intervalos de las dos mitades de una secuencia

Cruzamiento y mutación

Para la siguiente fase, se separó al primer cromosoma del resto. De esta manera este no será sometido a ningún ulterior proceso, conformando así siempre la primera lista de alturas generadas. Los otros cinco cromosomas (recordar que se están escogiendo los seis primeros) serán los que se cruzarán y mutarán.

Lo primero que se hace es combinar estas listas de a dos miembros de todas las maneras posibles (que serán $2N$ combinaciones), siendo N la cantidad de listas que se tengan (que

en este caso serán siempre cinco). Es decir, si se tiene [B C D E F] se obtendría [B C] [B D] [B E] [B F] [C D] [C E] [C F] [D E] [D F] [E F].

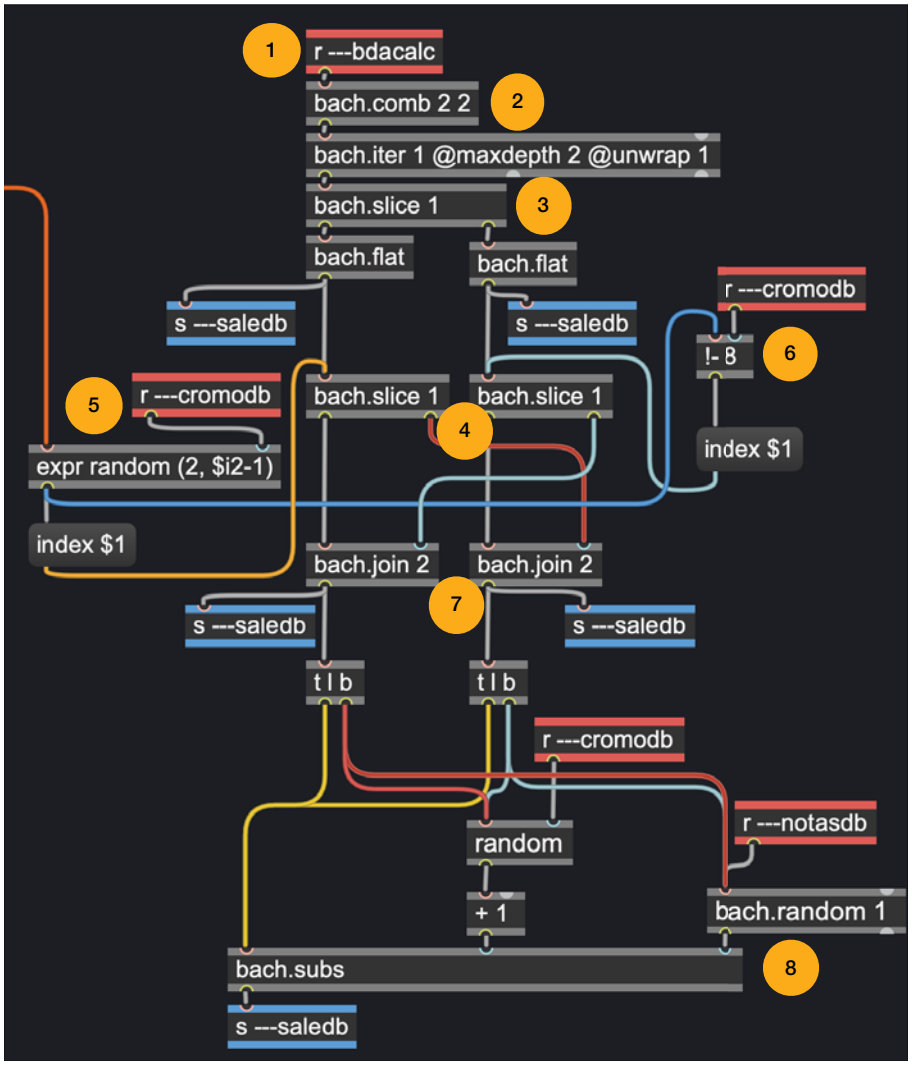


Figura 159: implementación del algoritmo de cruzamiento y mutación

En la figura superior se observa la implementación del algoritmo de cruzamiento y mutación. Los cinco cromosomas generados que se mencionaron llegan aquí a través de **r —bdacalc** (1) y son combinados en **bach.comb 2 2** de la manera que se vio arriba. Luego se iteran en **bach.iter 1 @maxdepth 2 @unwrap 1** y se separan en dos utilizando **bach.slice 1**. De esta manera, por ejemplo, la combinación [B C], es decir [[6000 6300

6600 6100 6800 6500 6400 7000] [7000 6100 6500 6000 6400 6600 6800 6300]] se separa en B y C. Estas listas ingresan a sendos **bach.slice 1** (4) que las dividen en dos con una cantidad de miembros complementarias y aleatorias. Es decir, si B se divide en dos grupos de 3 y 5 miembros, C se dividirá en dos grupos de 5 y 3 miembros. El punto aleatorio de separación lo da **expr random (2, $i2-1)** (5) que arroja valores aleatorios entre 2 y el tamaño del cromosoma menos uno. Este valor aleatorio ingresa al mensaje **index $1** que le indicará, al **bach.slice 1** de la izquierda, el punto de separación mientras que el mismo valor aleatorio ingresa a **!- 8** (6) para así obtener el valor complementario e indicarle al **bach.slice 1** de la derecha el punto de corte. Entonces si **expr random (2, $i2-1)** arroja un 5, por ejemplo, la lista B se dividirá en [[6000 6300 6600 6100 6800] [6500 6400 7000]] y la lista C en [[7000 6100 6500] [6000 6400 6600 6800 6300]]. Luego estas dos listas son cruzadas utilizando los dos objeto **bach.join 2** (7) de la siguiente manera:

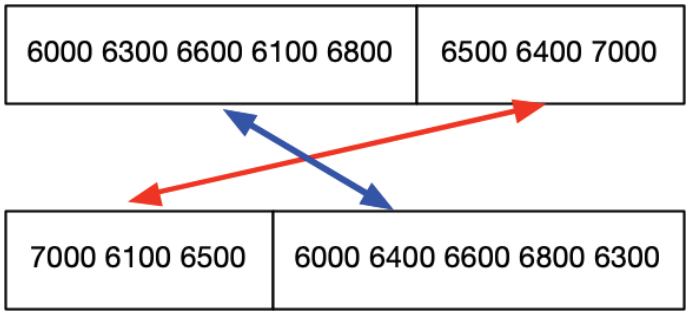


Figura 160: cruzamiento

Así se obtienen dos nuevas listas [[6000 6300 6600 6100 6800] [7000 6100 6500]] y [[6500 6400 7000] [6000 6400 6600 6800 6300]]. Lo mismo sucede con las diez combinaciones que generan en **bach.comb 2 2**.

Por último estas dos listas son mutadas en la sección indicada con un (8) en la figura 159. A continuación un detalle de esto:

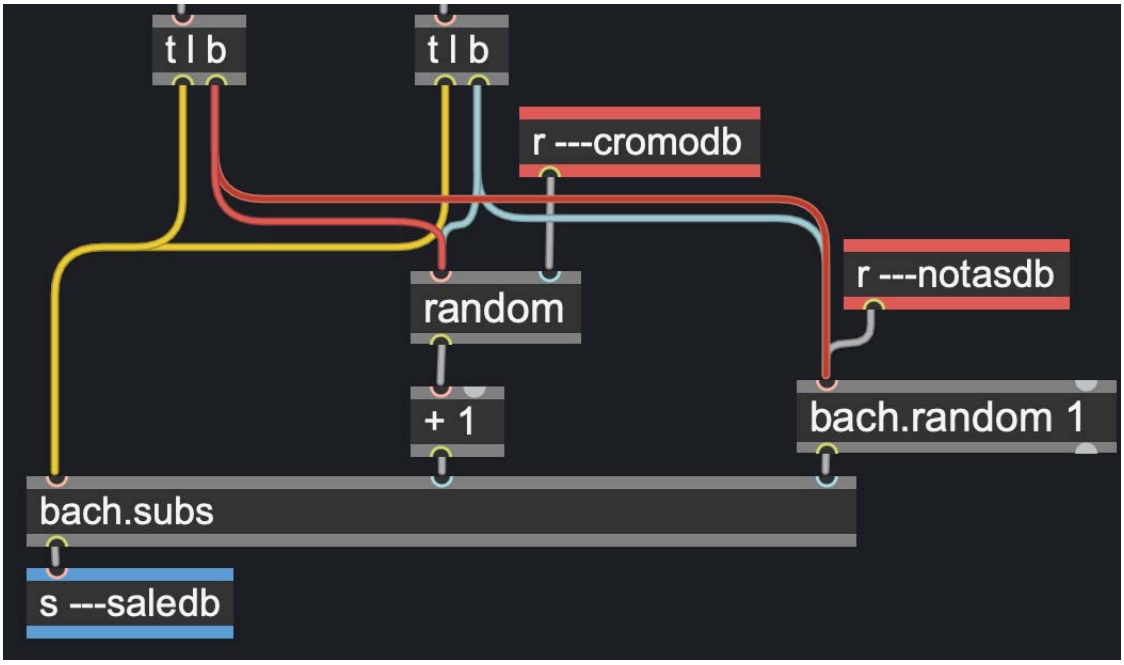


Figura 161: detalle del proceso de mutación

Cada una de estas dos nuevas listas obtenidas llegan a sendos objetos **t l b** que primero envían un **bang** y luego la lista que les ingresa. Por como *Max* maneja el orden de los mensajes (de derecha a izquierda), primero llegará la lista que ingresa por el **t l b** de la derecha y luego el de la izquierda. En ambos casos, cuando el **bang** sale de dicho objeto, escoge un valor al azar de las alturas seleccionadas en el inicio del algoritmo, esto se produce en **bach.random 1.** Las alturas seleccionadas en el módulo generador de alturas llega a través del objeto **r -—notasdb**. Esta nota sustituirá a alguna de las notas contenidas en la lista cruzada. Qué nota será la reemplazada depende del objeto **random** que indica una posición de sustitución al azar. Finalmente, todas estas listas, las cinco originales, las veinte cruzadas y las veinte mutadas serán almacenadas en una

segunda base de datos para que sean calculadas sus aptitudes nuevamente.

Segundo cálculo de aptitud

El segundo cálculo de aptitud que seleccionará las alturas que se van a mostrar al usuario se produce de una manera muy similar al primer cálculo de aptitud. Es decir que el sistema vuelve a analizar la desviación estándar y el IcVSIM pero esta vez de los 45 grupos de alturas resultantes del proceso de cruzamiento y mutación.

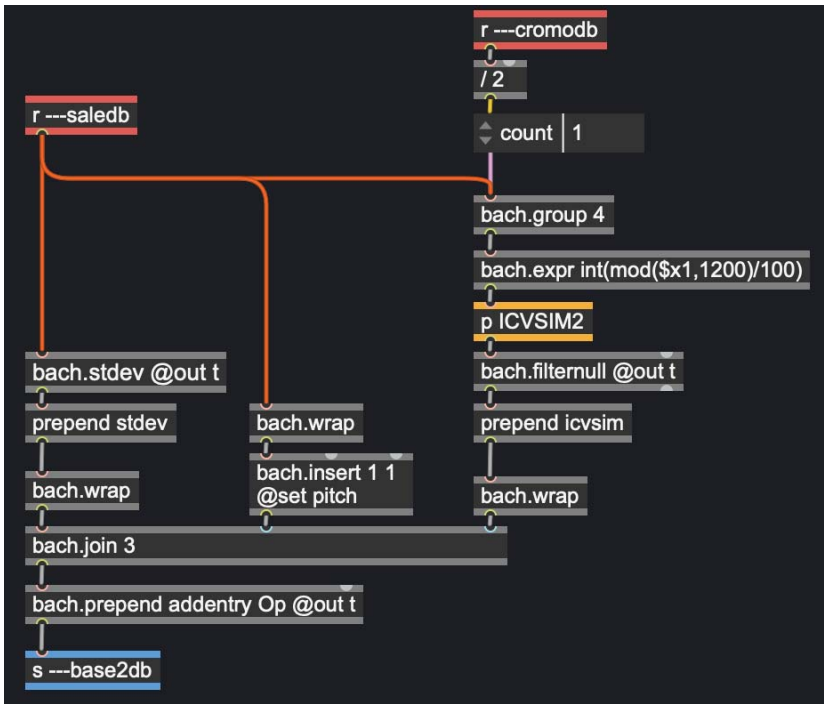


Figura 162: segundo calculo de aptitud

En la figura 162 es posible constatar que se están utilizando los mismos objetos que se usaron para la primera selección, ese decir, **bach.stdev** y **p ICVSIM2**. Este nuevo análisis se almacena en una segunda base de datos utilizando un objeto **dada.base op** desde donde se seleccionarán las secuencias finales.

Generación. De 45 cromosomas

Id	Alturas				IcVSIM
1	7000	6100	6300	6500	0.57735
2	7000	7000	6300	6500	0.372678
3	6500	6800	6600	6800	0.57735
4	6500	6800	6600	6400	0.57735
5	7000	6100	6800	6400	0.57735
6	7000	6100	6000	6400	0.57735
7	6500	6800	6100	6500	0.57735
8	6500	6800	6100	6500	0.57735
9	7000	6100	6800	6500	0
10	7000	6100	6800	7000	0.57735
11	6500	6800	6600	6100	0.57735
12	6500	6800	7000	6100	0
13	7000	6100	6300	6500	0.57735
14	7000	7000	6300	6500	0.372678
15	6500	6400	6600	6800	0.57735
16	6500	6800	6600	6800	0.57735
17	6300	6500	6800	6400	0.57735
18	6100	6500	6800	6400	0
19	6600	6100	6100	6500	0.57735
20	6600	6400	6100	6500	0.57735
21	6300	6500	6800	6500	0.57735
22	6300	6500	6400	6500	0.57735
23	6500	6400	7000	6100	0.57735
24	6600	6400	7000	6100	0.57735
25	6300	6500	6300	6500	0
26	6300	6600	6300	6500	0.57735
27	6600	6400	6300	6800	0.57735
28	6600	6400	6600	6800	0
29	6800	6400	6800	6500	0.57735
30	6800	6400	6600	6500	0.57735
31	6400	6500	7000	6100	0.57735
32	6100	6500	7000	6100	0.57735
33	6800	6400	6300	6500	0.57735
34	6800	6400	6300	6500	0.57735
35	6100	6500	6600	6500	0.57735
36	6100	6500	6600	6800	0.57735
37	6800	6500	6300	6500	0.57735
38	6300	6500	6300	6500	0
39	7000	6100	6600	7000	0.57735
40	7000	6100	6600	6800	0.57735
41	6500	6800	7000	6100	0
42	6600	6400	6300	6500	0
43	6100	6500	6800	6400	0
44	7000	6100	6800	6500	0
45	6600	6800	6300	6500	0

Tabla 5: ejemplo de datos dentro de **dada.base**

Esta selección, si bien se realizará, como en el caso anterior, a partir de escoger las secuencias considerando su desviación estándar o su coeficiente de IcVSIM, dentro de este universo se hará una selección de secuencias al azar con una probabilidad de tipo exponencial sin repetición. Por ejemplo, al generar cromosomas de 4 alturas, se acumula dentro de la base de datos lo que se observa en la tabla 5. Como se ve, algunos cromosomas poseen un IcVSIM de 0 (hay once de ellos), otros tienen un valor de 0.372678 (hay dos) y el resto tienen un coeficiente de 0.57735. A continuación el sistema escoge los primero dieciséis cromosomas que poseen los menores coeficientes. Entonces quedará como se ve en la Tabla 6.

Los primeros dieciséis cromosomas

Id	Alturas				IcVSIM
9	7000	6100	6800	6500	0
12	6500	6800	7000	6100	0
18	6100	6500	6800	6400	0
25	6300	6500	6300	6500	0
28	6600	6400	6600	6800	0
38	6300	6500	6300	6500	0
41	6500	6800	7000	6100	0
42	6600	6400	6300	6500	0
43	6100	6500	6800	6400	0
44	7000	6100	6800	6500	0
45	6600	6800	6300	6500	0
2	7000	7000	6300	6500	0.372678
14	7000	7000	6300	6500	0.372678
1	7000	6100	6300	6500	0.57735
3	6500	6800	6600	6800	0.57735
4	6500	6800	6600	6400	0.57735

Tabla 6: cromosomas escogidos

Aquí el sistema escoge, entonces, los dieciséis cromosomas mencionados. Sin embargo estos no saldrán en orden ascendente sino que serán seleccionados al azar, sin repetición, donde el peso de la probabilidad de ser escogido de cada uno es menor a medida que aumenta la posición en la tabla. Entonces el primer cromosoma (id 9) tendrá más probabilidad de aparecer que el id 4. Este peso de probabilidad esta distribuido de

manera exponencial (ver figura 163) para acentuar aún más la tendencia a la aparición de los cromosomas de los primeros puestos. Esta es en realidad la función del mensaje “GEN” que se observa en la figura 165, es decir, la de generar siempre esta lista aleatoria y escoger siempre un orden diferente dentro de los dieciséis cromosomas más aptos.

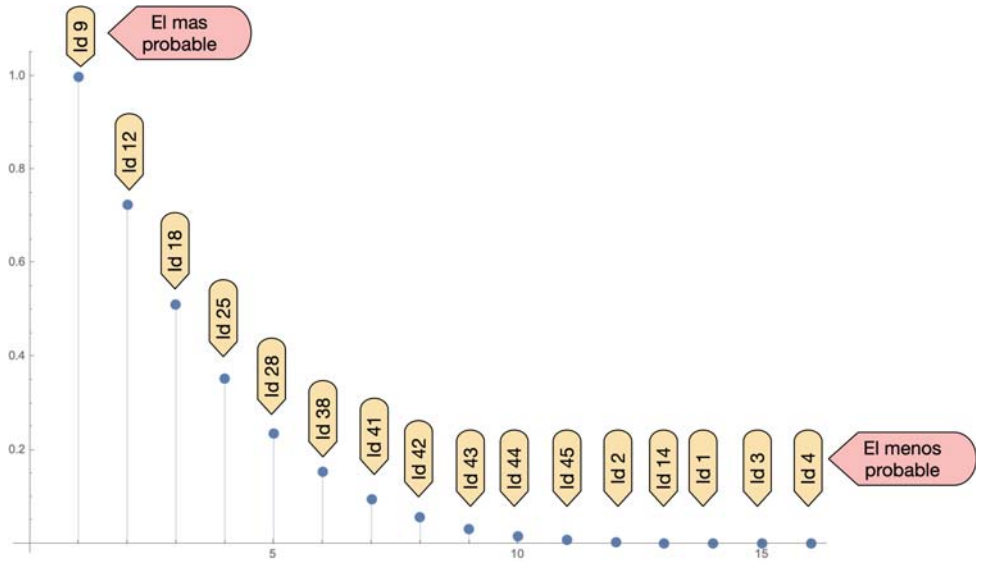


Figura 163: ordenados según su aptitud

Por ejemplo, una de estas generaciones arrojó la siguiente lista:

Ejemplo de generación

Id	Alturas				IcVSIM
18	6100	6500	6800	6400	0
43	6100	6500	6800	6400	0
9	7000	6100	6800	6500	0
25	6300	6500	6300	6500	0
12	6500	6800	7000	6100	0
28	6600	6400	6600	6800	0
41	6500	6800	7000	6100	0
38	6300	6500	6300	6500	0
44	7000	6100	6800	6500	0
2	7000	7000	6300	6500	0.372678
42	6600	6400	6300	6500	0
45	6600	6800	6300	6500	0
14	7000	7000	6300	6500	0.372678
1	7000	6100	6300	6500	0.57735
3	6500	6800	6600	6800	0.57735
4	6500	6800	6600	6400	0.57735

Tabla 7: ejemplo de generación

Aquí se pueden ver y escuchar las notas generadas que utilizan la tabla superior:

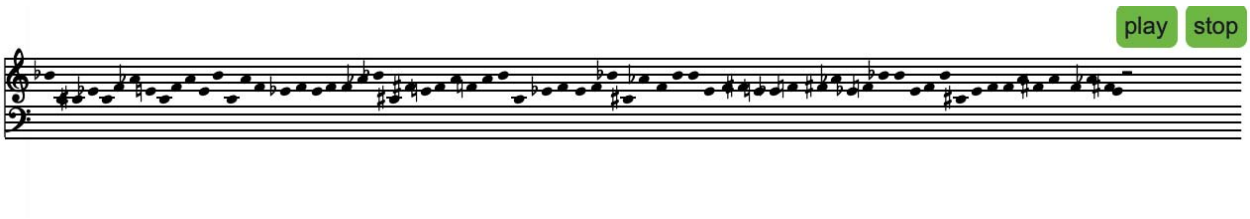


Figura 164: notas de la tabla 7

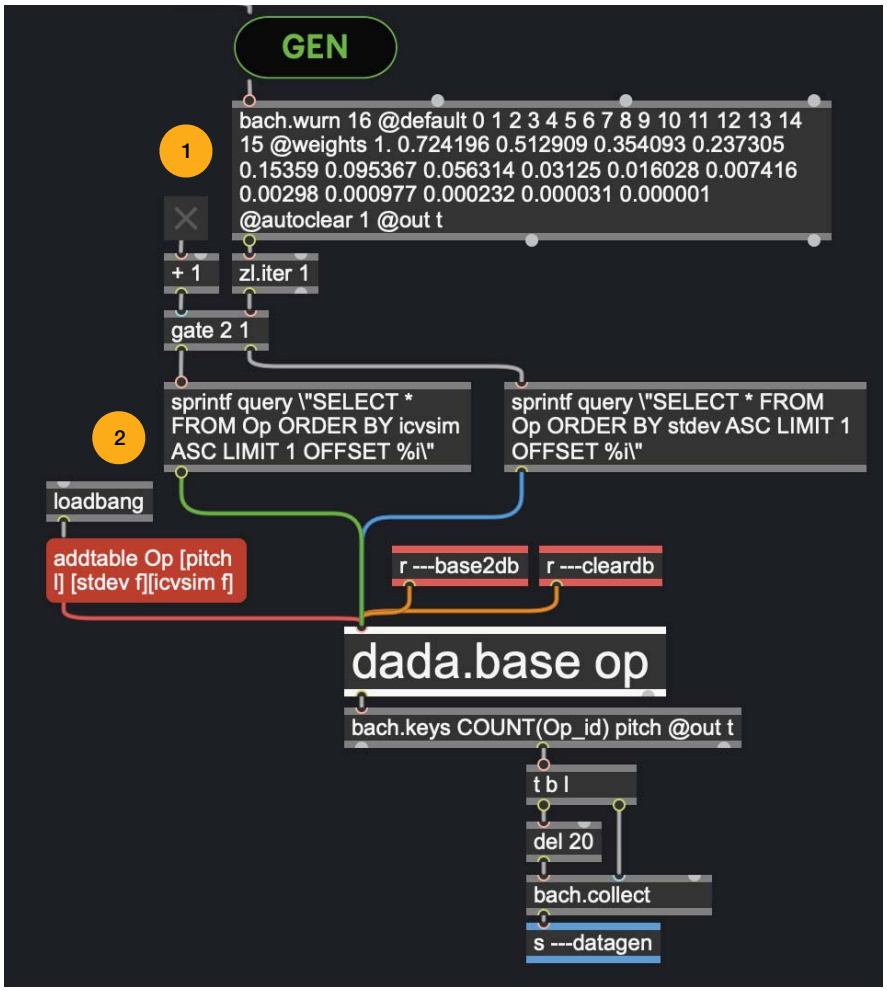


Figura 165: implementación de la selección final de los cromosomas

La implementación de este algoritmo comienza con el generador aleatorio que permite obtener valores sin repetición y con un probabilidad de aparición determinada. Para esto

es que se usa el objeto **bach.wurn** (1). Como se aprecia, dicho objeto tiene una gran cantidad de atributos declarados que especifican qué valores debe seleccionar y qué peso le debe dar a cada valor. Una vez generados estos valores, es posible determinar si estos ingresarán a uno u otro objeto **sprintf** (2) que serán los encargados de solicitar a la base de datos que envíe su lista de alturas ordenadas, ya sea por su indice de desviación estándar o por su IcVSIM.

Autómata celular

En la herramienta *AMI* se ha implementado un sistema de *CA* de una dimensión ya que las de dos dimensiones (como es el caso del Game of Life) son más interesante y útiles para generar datos en tiempo real que para generar listas para ser almacenadas. El algoritmo de una dimensión que se ha diseñado utiliza, además, un vecindario de tres miembros[96] es decir que posee 256 posibles combinaciones.

El objeto principal utilizado para implementar este algoritmo es el mismo **cage.chain** usado para los *L-Systems*, sólo que en este caso generará una cadena de 20 células donde el tipo de reglas y el estado inicial son diferentes. Desde la interfaz específica para este algoritmo (figura 166) se puede ingresar el tipo de regla que se va a utilizar[97] (1), el estado inicial (2), las notas que serán usadas cuando la célula tenga un valor de 1 (3), el tamaño de la célula inicial (4), la cantidad de notas a generar (número amarillo con triángulo gris) y una serie de botones en color verde que permiten variar la secuencia de alturas.

96 Ver capítulo 2 sección 2.6.2

97 Implementada de la misma manera que se vio en la figura 92 del capítulo 2

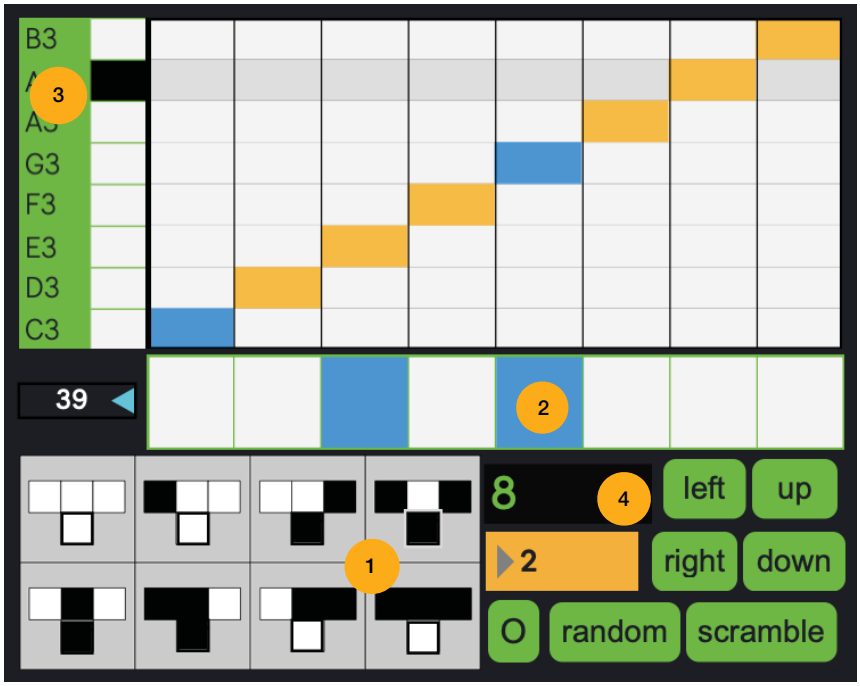


Figura 166: interfaz para el algoritmo de autómata celular

En el caso específico de la figura 166 se tiene, como se observa, la siguiente regla:

[0 0 0] → 0
[1 0 0] → 0
[0 0 1] → 1
[1 0 1] → 1
[0 1 0] → 1
[1 1 0] → 1
[0 1 1] → 0
[1 1 1] → 0

El estado inicial será [0 0 1 0 1 0 0 0]. Se debe considerar que el objeto **cage.chain** completa los faltantes de las puntas repitiendo el valor que se encuentra en una u otra posición. Como se explicó en el capítulo 2, existen diferentes técnicas para completar

estos valores y de hecho el objeto **cage.chain** puede implementar diferentes soluciones. Sin embargo en este caso se escogió su funcionamiento por defecto que es el que se menciona más arriba. Entonces el estado inicial [0 0 1 0 1 0 0 0] será completado así [0][0 0 1 0 1 0 0 0][0] ya que comienza y termina con 0. Se obtiene entonces:

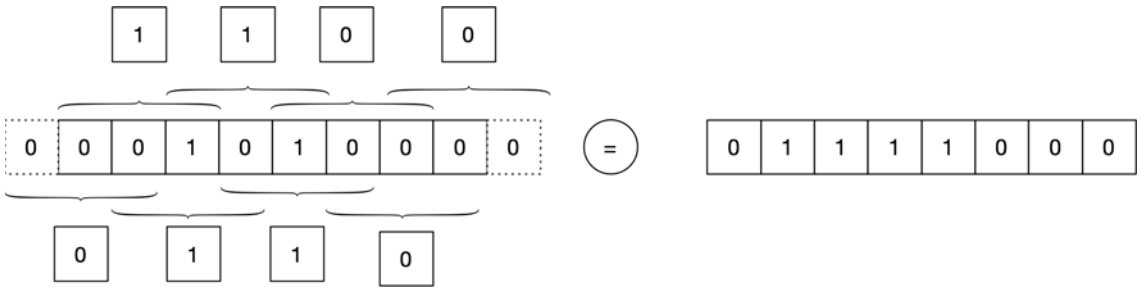


Figura 167: resultado

Después de 20 iteraciones queda:

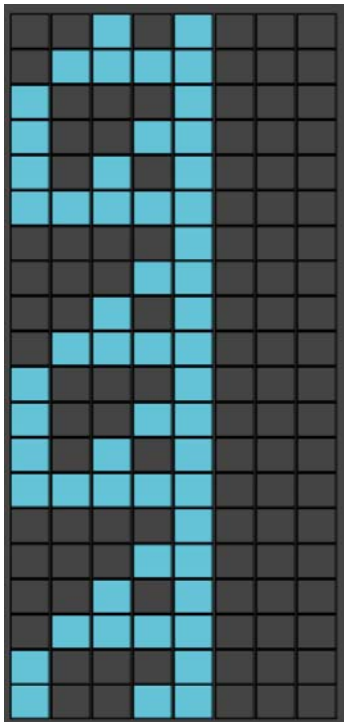


Figura 168: gráfico del resultado

Una vez generada la secuencia como la de la figura 168, cada casillero ocupado con un 1 activa la nota que se especifica en la interfaz (figura 166 (3)). Así, siempre siguiendo con el mismo ejemplo, en la figura 169 se ve la cadena de notas que generaría la secuencia de la figura superior.

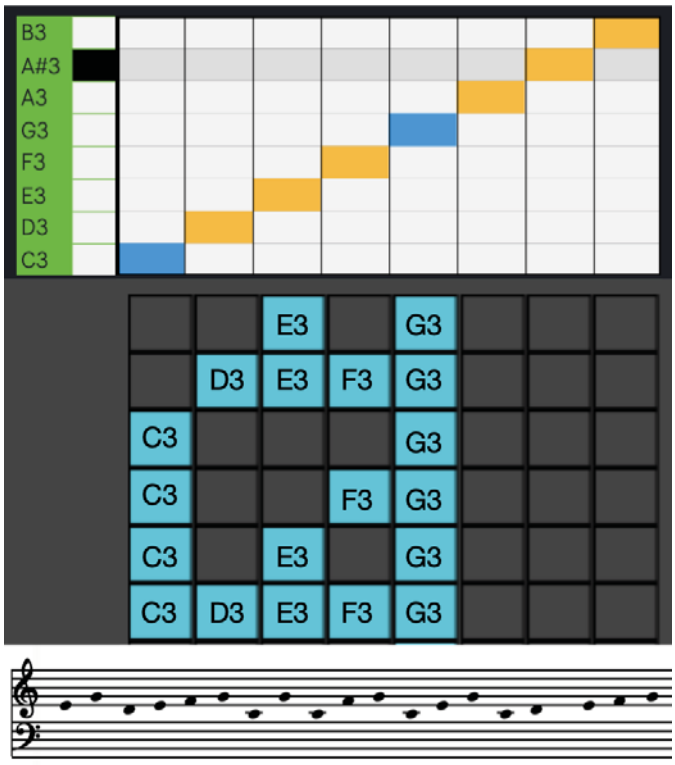


Figura 169: resultado de generado con las notas escogidas

Las notas disponibles y el ámbito de las mismas se especifica desde la misma interfaz que se ha utilizado hasta aquí para tal fin. También es posible modificar el orden y la secuencia de notas, siempre dentro de estas mismas notas escogidas, a partir de los botones “left”, “up”, “right”, “down”, “random” y “scramble”. En el siguiente video se observa un ejemplo de esto que se acaba de explicar.

Video 31: generación de alturas a partir de un autómata celular

3.3 - Generador de ritmos

La interfaz para la creación de ritmos también utiliza un generador de perfiles y un sector donde se puede especificar la manera de mapear dicho perfil. Aquí la gran diferencia con las alturas es que dicho mapeo no utiliza figuras rítmicas distintas para cada valor del perfil sino que será una cantidad de subdivisión de un valor unitario dado (redonda, blanca, negra, etc.). Se ha implementado este tipo de sistema para evitar las complejidades rítmicas que se suscitan si no se considera completar dichas unidades. Por ejemplo, si un perfil determinado arroja lo siguientes valores [1 0.5 1 0.33 0.5 0.25 0.25 0.2 0.33 0.5 1] y estos valores se escalan a partir de la negra donde esta vale 1, quedaría entonces una división rítmica como la que muestra la siguiente figura:

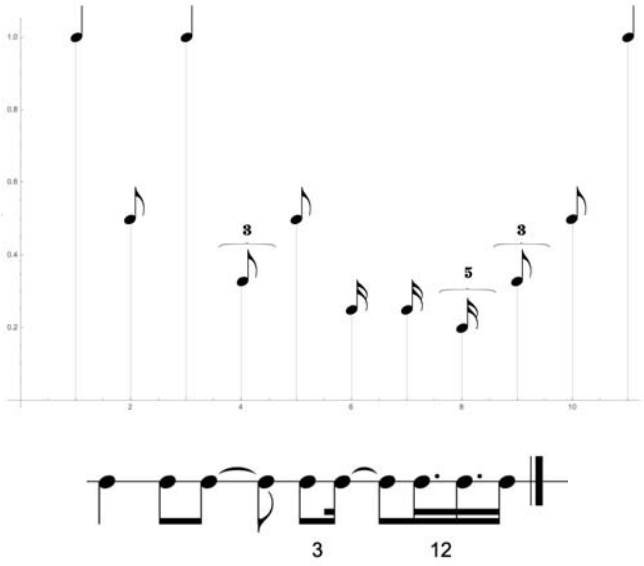


Figura 170: duración exacta de la figura musical

Como se aprecia, si bien la rítmica de la figura superior es válida y puede ser interpretada sin ningún problema por una computadora, se vuelve muy inapropiada para que la interprete un ejecutante humano y *AMI* es una herramienta que fue creada para que pueda ser utilizada por compositores de diferentes estilos y disciplinas, por lo que la legibilidad del resultado es importante. Lo que se ha implementado entonces, es un sistema donde la posición en el perfil indica la cantidad de subdivisiones de la unidad. De esta manera la partitura queda asi:

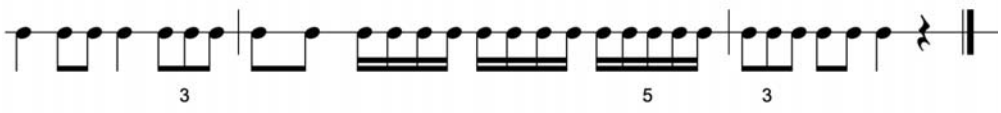


Figura 171: partitura resultante

Por supuesto que esta solución genera una mayor cantidad de notas pero estas serán acotadas a la cantidad seleccionada en la interfaz de alturas. Sin embargo esta es una de

las funcionalidades de la plataforma que serán expandidas y mejoradas en el futuro.

3.3.1 Módulo para generar ritmos

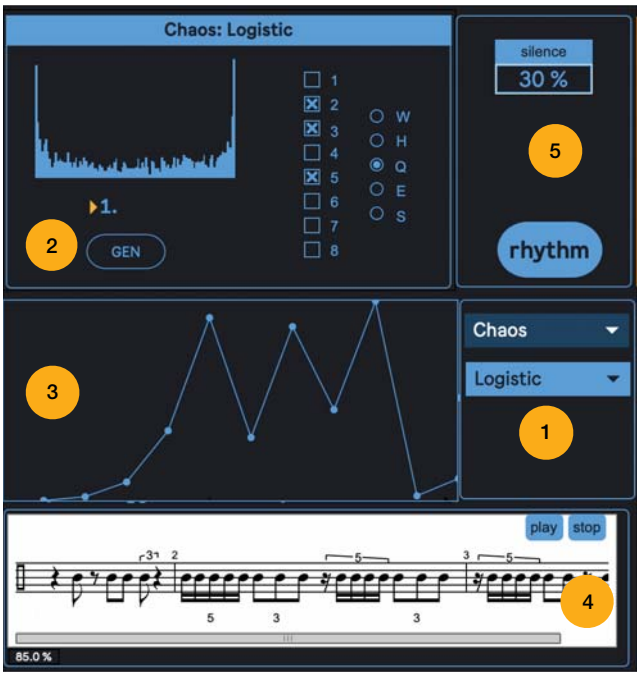


Figura 172: interfaz para generar ritmos.

La interfaz es muy similar a la de alturas. Existe un sector desde donde se escoge el generador de perfiles (1), otro desde donde se manipulan los parámetros del generador escogido (2), un sector donde se puede observar un gráfico de dicho perfil (3) y una partitura donde se aprecia el resultado de las manipulaciones (4). Hay, además, un sector especial desde donde se puede especificar una cantidad porcentual de silencios que el sistema generará (5).

3.3.2 Generador de perfiles

En cuanto la selección y manipulación de perfiles es exactamente igual a la de alturas, con la excepción de que aquí hay menos opciones de perfiles:

- L-System: algoritmo 1 y algoritmo 2.
- Caos: logístico, congruencial, sine.
- Aleatorios: random walk, lineal, triangular.
- Genéticos
- Autómata celular

Los generadores L-System, caos y aleatorios no necesitan mayor explicación ya que son idénticos a los ya expuestos, con la diferencia que la cadena de valores que generen será interpretada como divisiones de la unidad. La implementación del algoritmo para autómata celular, por otro lado, si bien también es muy similar a la utilizada anteriormente, difiere en que los valores de 1 y 0 generados por el objeto **cage.chain** serán interpretados, no como divisiones de la unidad sino como valores de duración (en el caso del 1), y de silencio para el 0. Así, la siguiente lista [0 1 0 1 0 0 0 1 1 0 1 0] será transformada en:

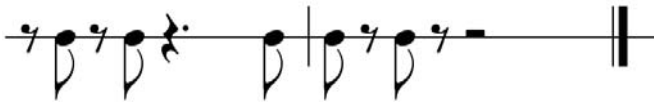


Figura 173: ritmo generado a partir de CA

Por lo tanto en el caso de este generador, el porcentaje de silencio que puede especificarse desde la interfaz que se muestra en la figura 172 (5) no es válido y de hecho se deshabilita cuando se escoge este generador.

Por último, la implementación del algoritmo genético para generar ritmos es el que más difiere del utilizado para las alturas. En este caso la generación no se produce a partir de

la selección de una serie de candidatos sino que se especifica una probabilidad o peso para escoger una serie de duraciones. Estas pueden ser: negra, corchea, corchea con puntillo, tresillo de corchea, semicorchea, quintillo y seisillo. A continuación se observa la interfaz de este generador.

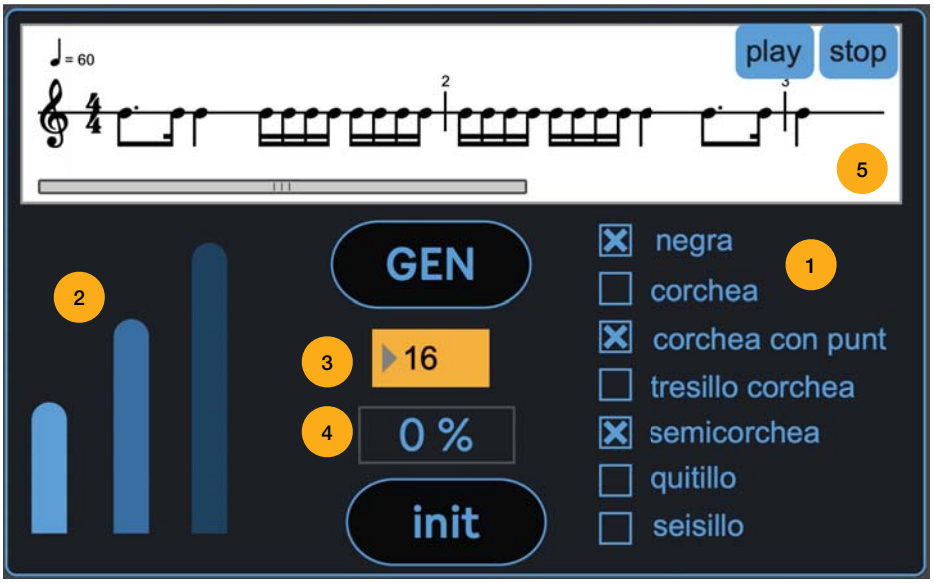


Figura 174: interfaz para la generación de ritmos a partir de una algoritmo genético

En la figura 174 se ve la interfaz mencionada. En el sector marcado con (1) es posible escoger una o varias de las figuras rítmicas que se mencionaron. Al seleccionar mas de una, se habilita un sector (2) desde donde se puede especificar la probabilidad de una u otra duración. En el ejemplo de la figura 174 la negra será la menos probable, luego la corchea con puntillo y por último la más probable de aparecer será la semicorchea. Es posible corroborar esto viendo las figuras rítmicas que aparecen en la partitura superior (5). Desde el número entero amarillo (3) se especifica el tamaño del cromosoma inicial (en unidades) y desde (4) la probabilidad que una duración sea reemplazada por un silencio. Cuando se presiona el botón “init” se crea este cromosoma, y al presionar “GEN” se genera la partitura. En el siguiente video se puede ver este funcionamiento.

Video 32: generación genética de ritmos

Como se puede observar, al seleccionar las duraciones de negra, corchea y semicorchea, se activan los faders que se ven en (2) y al seleccionar diferentes relaciones entre ellos y oprimir “init”, se generan cromosomas que tienen la chance de poseer más duraciones de un tipo que de otro. También se puede definir aquí el porcentaje de silencio que se desee.

Aquí, a diferencia de lo explicado en el algoritmo genético encargado de generar alturas, solo se aplicó un tipo de proceso de selección de los candidatos, y dichos seleccionados son los que formarán parte de la partitura final, es decir, que no se aplicó un proceso de cruzamiento y mutación. La selección se realiza comparando la similitud cronotónica (Hofmann-Engl, 2002) que posee el cromosoma original con respecto al resto y los más semejantes son seleccionados desde el de mayor al de menor similitud.

3.2.1 Similitud cronotónica

Existen diferentes investigaciones que se han ocupado de la similitud rítmica a nivel simbólico y el desarrollo de algoritmos que puedan medirla, aunque estas son menos frecuentes que las propuestas para calcular la similitud melódica (Velardo, Vallati, & Jan, 2016) debido, entre otras muchas razones, a que esta última se utiliza en gran medida para detectar infracciones al copyright de una obra (Heo, Kim, Kim, & Lee, 2017; Park, Kwon, Lee, Kim, & Nam, 2019; Schuitemaker, 2020).

Dentro de los trabajos que abordan la cuestión rítmica es posible citar investigaciones enfocadas mayormente a la cuestión cognitiva, es decir, que requieren, para la búsqueda de similitudes, tomar en consideración la percepción rítmica (Gómez Marín, Jordà Puig, & Boyer, 2015) y para ello acotan su campo de trabajo a ritmos en 4/4 con una extensión de un compás y una subdivisión de semicorcheas (Lerdahl & Jackendoff, 1983; Post & Toussaint, 2011). Otras propuestas, quizá de modo opuesto, utilizan la distancia de Levenshtein (Levenshtein, 1966) para calcular la cantidad de cambios, sustituciones y omisiones que deben implementarse en una cadena de símbolos para convertirla en otra de referencia (Moritz, Heard, Kim, & Lee, 2020; Post & Toussaint, 2011). Este tipo de medida será utilizada en la presente herramienta para analizar las duraciones, alturas y dinámicas generadas como una opción más para encontrar similitudes, pero tiene la desventaja de no considerar los aspectos cognitivos de la rítmica. Al implementar este algoritmo genético, se ha utilizado la similitud cronotónica ya que, como su autor manifiesta, intenta “construir un modelo de similitud rítmica que se pondrá a prueba dentro de un experimento con el intención de cerrar la brecha entre la ciencia cognitiva y la comunidad de modelos.” (Hofmann-Engl, 2002).

El procedimiento es el siguiente: primero se debe encontrar el máximo común divisor (MCD) de las duraciones que se van a utilizar. Si se trabaja con semicorcheas, corcheas y negras esta sería la semicorchea (1/16), pero como la presente implementación utilizará, como ya se dijo, negras, corcheas, corcheas con puntillo, tresillos de corchea, semicorcheas, quintillos y seisillos, el MCD será 1/240. Luego hay que convertir el ritmo de referencia (sin importar si las duraciones corresponden a silencios o no) en una cadena de beats atómicos que consistirá en colocar tantos valores N como N valores quepan dentro de la figura. Por ejemplo, en una semicorchea (1/16) caben 15 átomos de 1/240, en un silencio de tresillo de corchea caben 20, etc. En la siguiente figura se observa, sobre cada figura rítmica, la cantidad de átomos que caben en cada una.

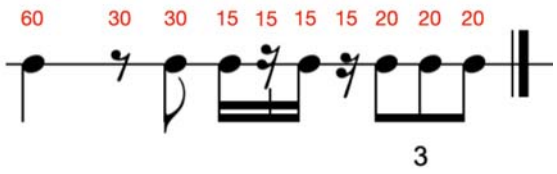


Figura 175: ejemplo con sus correspondientes átomos

Entonces la cadena de beats atómicos de este ritmo será: [60 60 60 60 60 60…60 veces, 30 30 30 30…30 veces, 30 30…30 veces, 15 15…15 veces, etc.]

A continuación vemos el ritmo que se comparará con el primero:

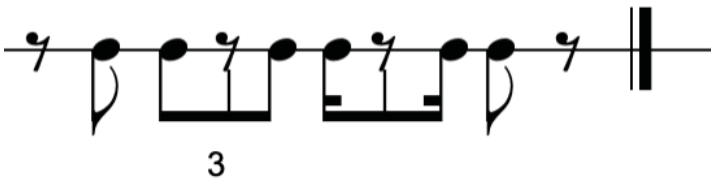


Figura 176: segundo ejemplo

Su cadena es: [30…30 veces, 30…30 veces, 20…20 veces, 20…20 veces…etc]

Se puede ver en la figura 177 un gráfico del ritmo *A* con sus puntos en color rojo y otro del ritmo *B* con sus puntos en azul. Luego se ve un gráfico de ambos superpuestos. Si las cadenas poseen diferente tamaño se deben ecualizar, esto es, multiplicar a una u otra por una constante que las haga iguales. El siguiente paso es superponer una cadena sobe la otra pero con una de ellas invertida en su signo y sumarlas. En la figura 178 se ve un gráfico de este proceso.

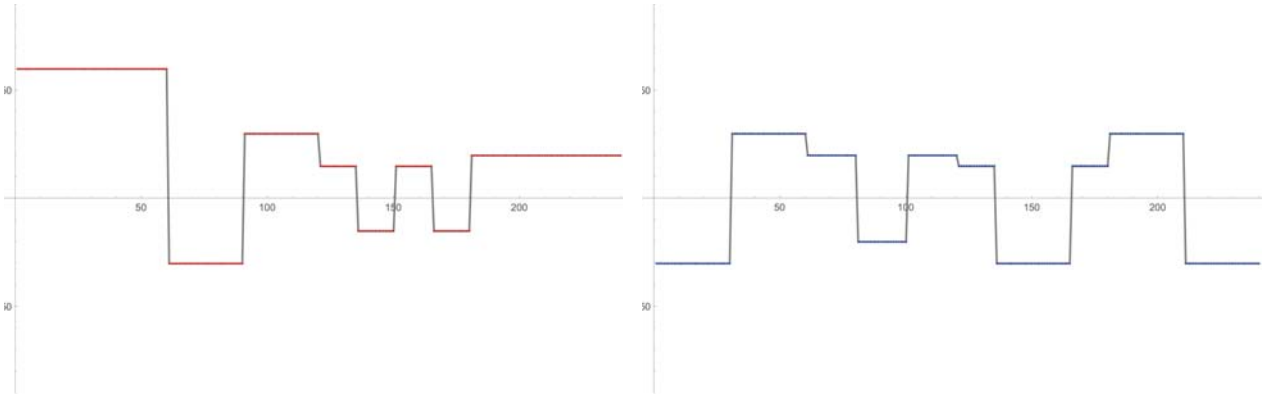


Figura 177: ritmo A (izquierda) y ritmo B (derecha)

Cuanto menor es la diferencia entre estas dos cadenas, menor es la diferencia rítmica. De hecho es posible constatar en la figura 178, que alrededor de N = 120 la diferencia es de 0, que es donde ambos ritmos coinciden. La ecuación para calcular la similitud queda:

$$\sqrt{\frac{\sum_{i=1}^{n}\left(e^{-\frac{k_2}{n}r_i^2}\right)^2}{n}}$$

Ecuación 34

Donde n es el tamaño de la cadena en beats atómicos (después de ser ecualizada), r_i es el $i$ésimo componente de la cadena reflejada y k_2 es una constante empírica. En el caso de la figura 178 el valor que arroja la ecuación 34 es 0.42525.

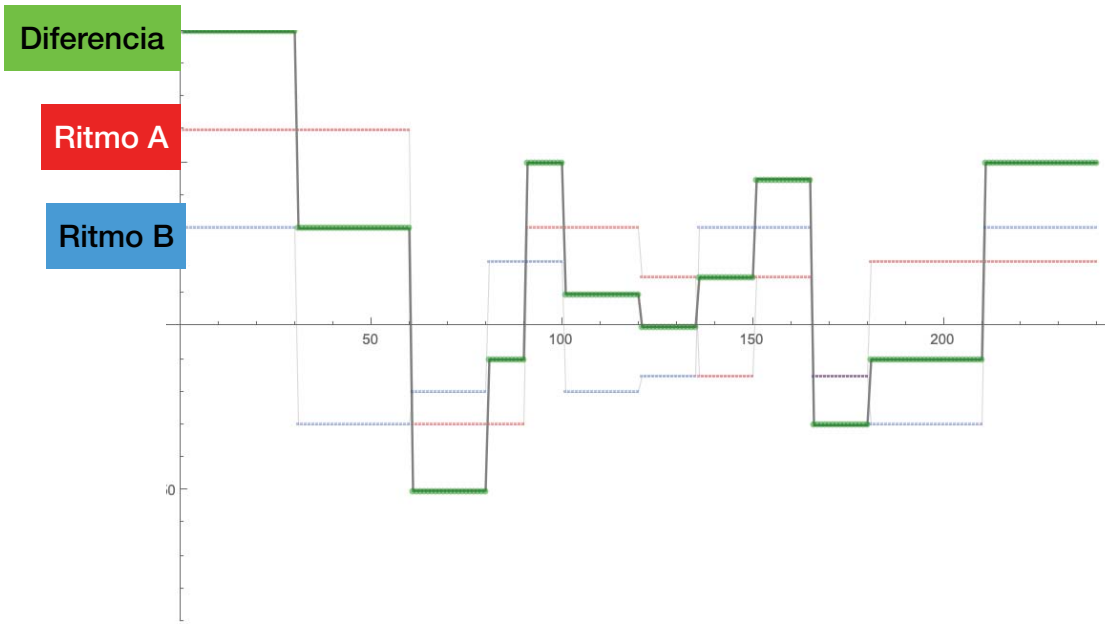


Figura 178: gráfico de ritmos superpuestos y su diferencia

3.2.2 Implementación en Max

La generación de datos comienza con un objeto **bach.wrandom** que generará valores aleatorios a partir de una cadena de números dada. Dicha lista la especifica el usuario a través de la interfaz que se ve en la figura 174 (1). Así, si se escoge por ejemplo utilizar negras, tresillos y semicorcheas, la lista desde donde **bach.wrandom** escogerá sus valores será [1 4 5]. Entonces si le pide que elija ocho valores, se puede obtener por ejemplo [1 4 1 5 5 4 1 4]. Esta lista ingresará a un objeto especial de la librería *Bach* que se llama **bach.eval** y que permite utilizar dentro de él un lenguaje de programación

propio llamado *bell* (bach evaluation language for ||||s[98]). Este objeto se utiliza en varias partes de la programación de la herramienta *AMI*. En este caso específico lo que hace es interpretar la lista aquí mencionada como una serie de direcciones que apuntarán a figuras rítmicas específicas y a combinaciones de las mismas.

En la figura 179 se ve una referencia de estos ritmos y sus direcciones. La lista [1 4 1 5 5 4 1 4] escogerá, entonces, una negra, uno de los tres ritmos (seleccionados de manera aleatoria) de tresillo de corchea, otra negra, uno de los cuatro ritmos de semicorchea, etc.

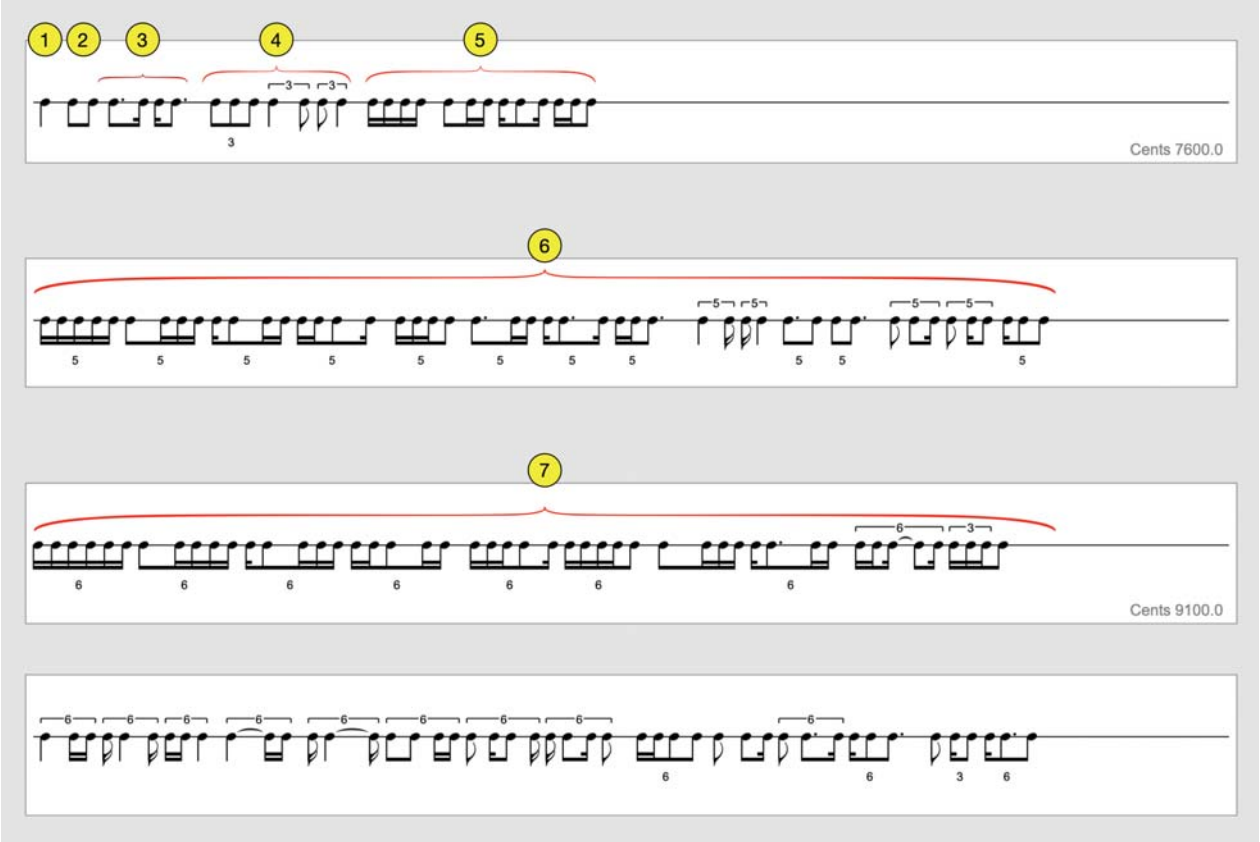


Figura 179: ritmos posibles para los cromosomas

[98] Acrónimo de Lisp-like linked list

A continuación se observa la programación dentro de **bach.eval** que permite esto.

```
$ri1 = 1/4;
$ri2 = 1/8 1/8;
$ri3 = 3/16 1/16;
$ri4 = 1/12 1/12 1/12;
$ri5 = 2/12 1/12;
$ri6 = 1/16 1/16 1/16 1/16;
$ri7 = 1/16 1/16 1/8;
$ri8 = 1/20 1/20 1/20 1/20 1/20;
$ri9 = 2/20 1/20 1/20 1/20;
$ri10 = 3/20 1/20 1/20;
$ri11 = 4/20 1/20;
$ri12 = 2/20 2/20 1/20;
$ri13 = 2/20 3/20;
$ri14 = 1/24 1/24 1/24 1/24 1/24 1/24;
$ri15 = 2/24 1/24 1/24 1/24 1/24;
$ri16 = 3/24 1/24 1/24 1/24;
$ri17 = 4/24 1/24 1/24;
$ri18 = 5/24 1/24;
$ri19 = 2/24 2/24 1/24 1/24;
$ri20 = 3/24 2/24 1/24;

$in = $x1;

$sal1 = if $in == 1 then $ri1
else if $in == 2 then $ri2
else if $in == 3 then scramble($ri3)
else if $in == 4 then scramble([$ri4][scramble($ri5)]):1
else if $in == 5 then scramble([$ri6][scramble($ri7)]):1
else if $in == 6 then scramble([$ri8][scramble($ri9)] [scramble($ri10)][scramble($ri11)]
[scramble($ri12)][scramble($ri13)]):1
else if $in == 7 then scramble([$ri14][scramble($ri15)][scramble($ri16)]
[scramble($ri17)][scramble($ri18)][scramble($ri19)][scramble($ri20)]):1;

$sal1
```

Una vez escogidos los ritmos dentro de **bach.eval** estos ingresan a un sector de la programación donde algunas de las duraciones se convierten en silencios a partir de otro proceso aleatorio donde el usuario estipula (figura 174 (4)) el porcentaje de probabilidad de que esto ocurra. Todo esto se produce cuando el usuario presiona el botón “init”, el que podrá presionar todas la veces que quiera hasta que el ritmo del cromosoma original sea de su agrado.

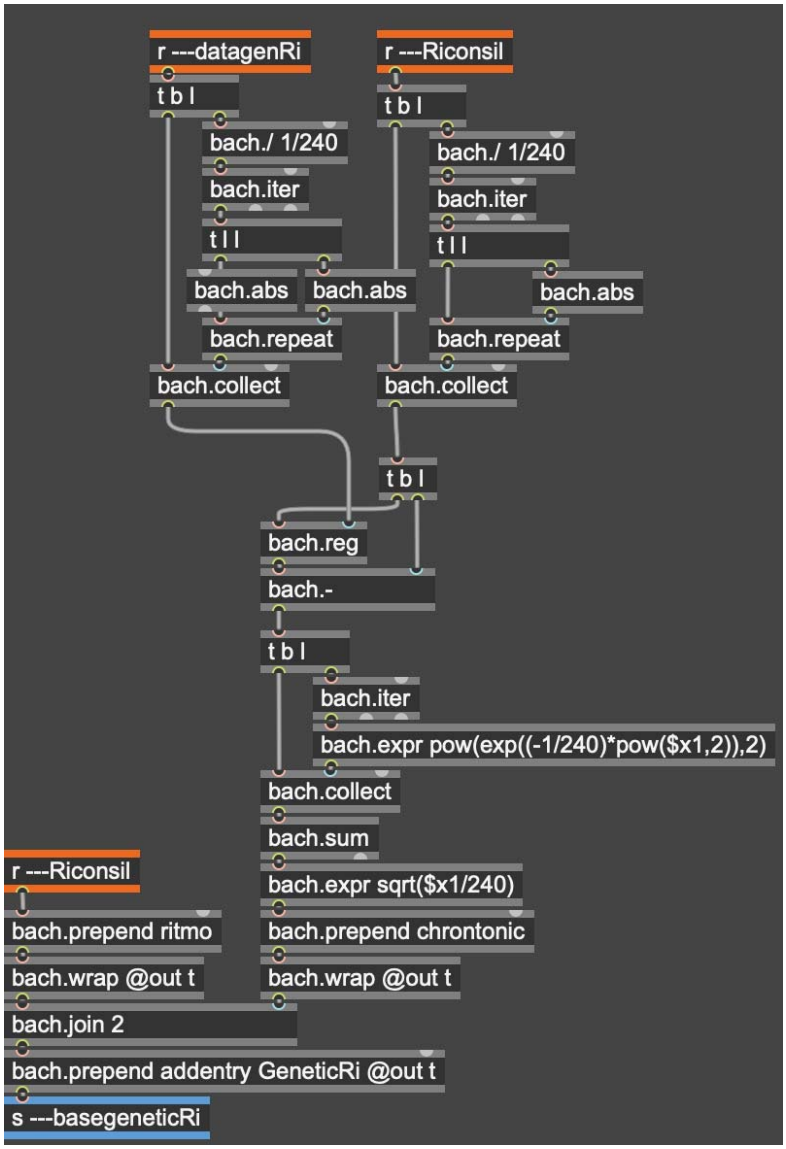


Figura 180: implementación del cálculo de la similitud cronotónica

Luego, el primer cromosoma es comparado con otros veinte generados por el sistema cuando el usuario presiona el botón “GEN”. Aquí entonces se produce el cálculo del índice de similitud cronotónica que posee cada generación con el cromosoma original. Este cálculo se produce en la programación que se ve en la figura 180.

Los ritmos y su índice cronotónico son almacenados una vez más dentro de un objeto **dada.base** que arrojará a continuación dichas listas ordenadas de manera ascendente desde el ritmo con mayor similitud al de menor. Este proceso se produce en la programación que se observa en la figura 181.

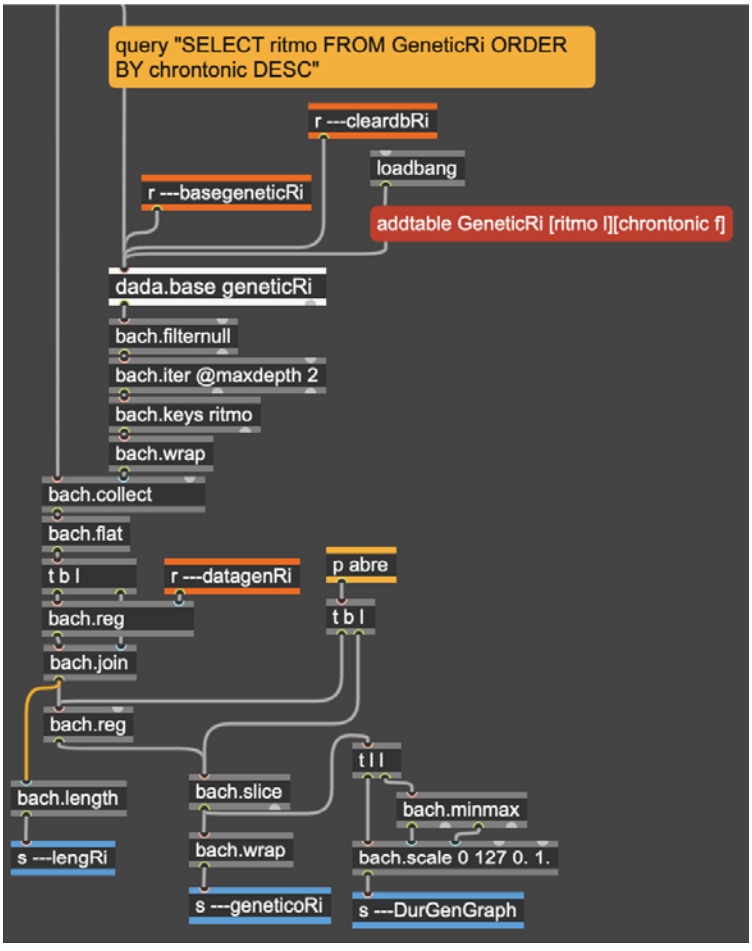


Figura 181: almacenamiento y ordenamiento de los ritmos generados

Así entonces, cuando se genera un cromosoma con el botón “init”, con las duraciones según sus probabilidades y con la cantidad de silencio especificado, se generarán aleatoriamente, al presionar el botón “GEN”, una serie se cromosomas que utilizarán estos mismos parámetros y que se ordenarán de acuerdo a su similitud cronotónica.

3.4 - Generador de dinámicas

Esta interfaz, al igual que las anteriores utiliza algunos de los métodos de generación de perfiles que ya se han expuesto. Solo uno de ellos es totalmente nuevo ya que es muy apropiado para obtener patrones de dinámicas. Otra característica de este módulo es que es el más sencillo en cuanto a su mapeo. Simplemente escala los valores generados entre un mínimo y un máximo dado por el usuario.

3.4.1 - Módulo para generar dinámicas

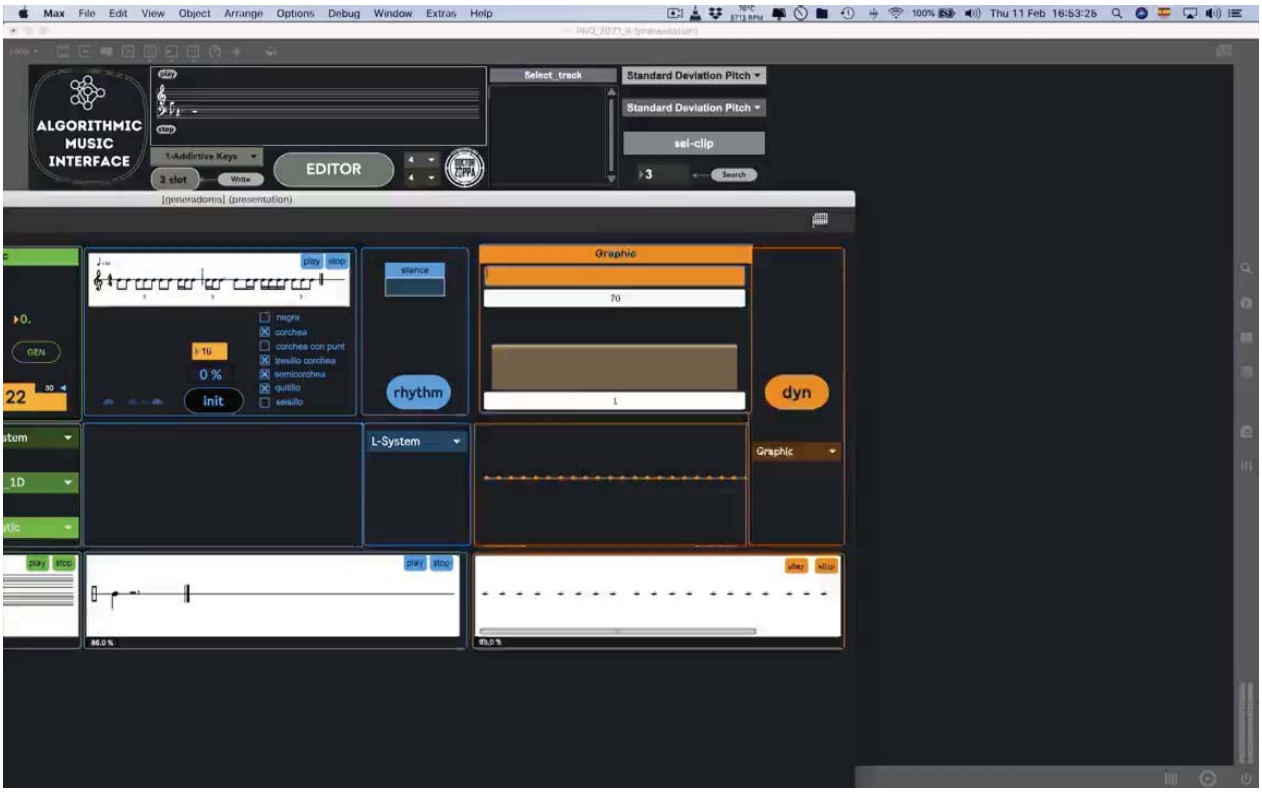
Los tipos de generadores utilizados aquí son (y que se escogen desde figura 182 (1)):

- L-System: algoritmo 1 y algoritmo 2.
- Caos: logístico, congruencial, sine.
- Aleatorios: random walk, lineal, triangular.
- Autómata celular
- Gráfico

Como se aprecia, el único módulo novedoso es el llamado *gráfico,* que será explicado a continuación.

Figura 182: interfaz para la generación de dinámicas

En la sección (2) se implementó, como ya se ha hecho hasta aquí, el sector desde donde se modificarán los parámetros de cada generador y en (3) es desde donde el usuario asigna un rango de valores donde el mínimo puede ser 0 y el máximo 127. Esto es válido para todos los generadores excepto el gráfico. Se aprecia en el video 33 el funcionamiento de la interfaz del módulo. En dicho video se observa que en la barra superior de color amarillo se pueden ingresar una serie de valores enteros que serán los grupos de notas que se verán afectadas por el mismo valor de dinámica. Entonces si se escribe [2 3 4 5 3] se estará estipulando que la secuencia podrá modificarse cada 2, 3, 4, 5 y 3 notas. Cuando se hace esto aparecen sendos faders desde donde se pueden especificar el valor de key velocity de la nota. En el ejemplo del video 33 se hubiera obtenido algo como lo que se ve en la figura 183.



Video 33: funcionamiento de la interfaz gráfica

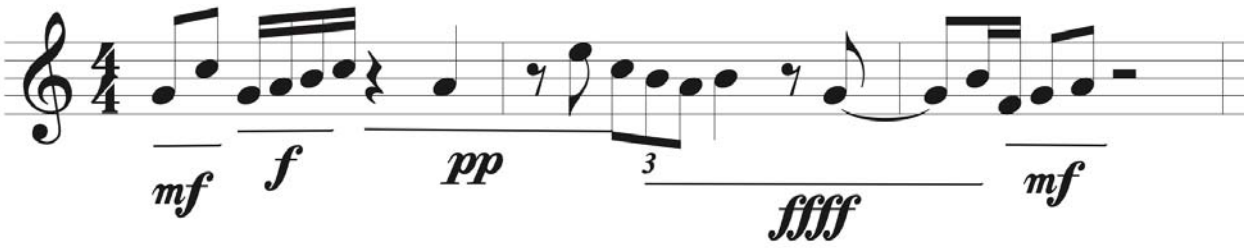
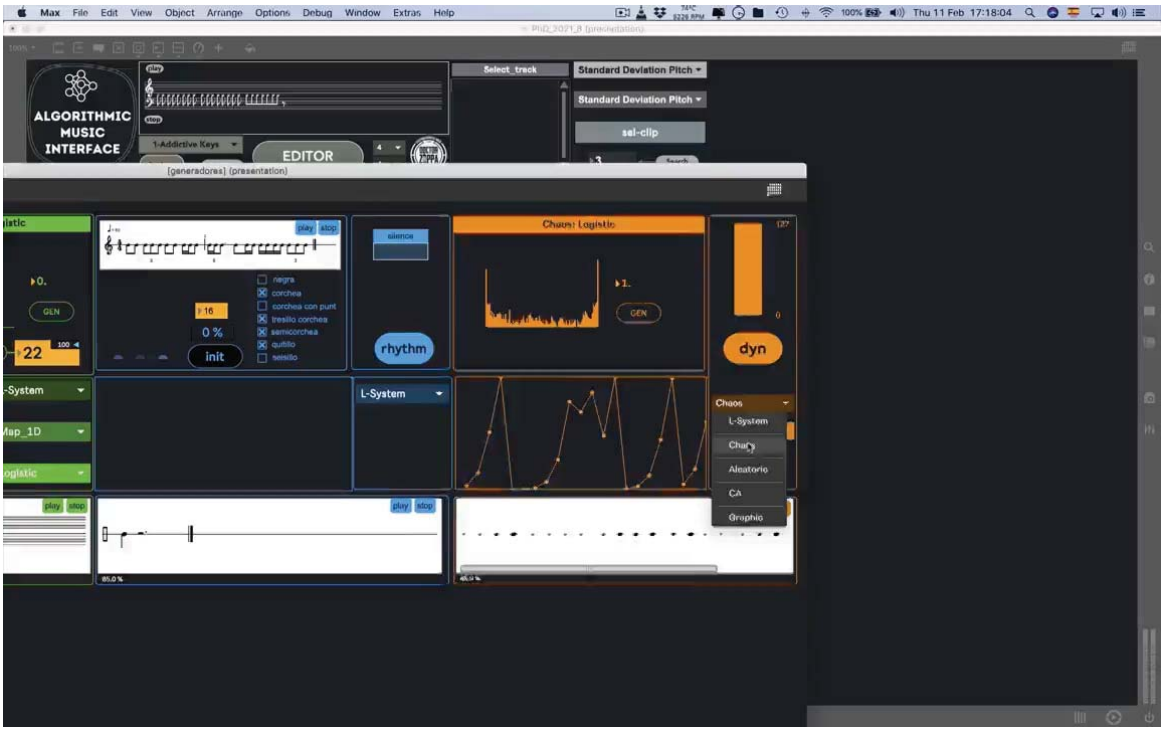


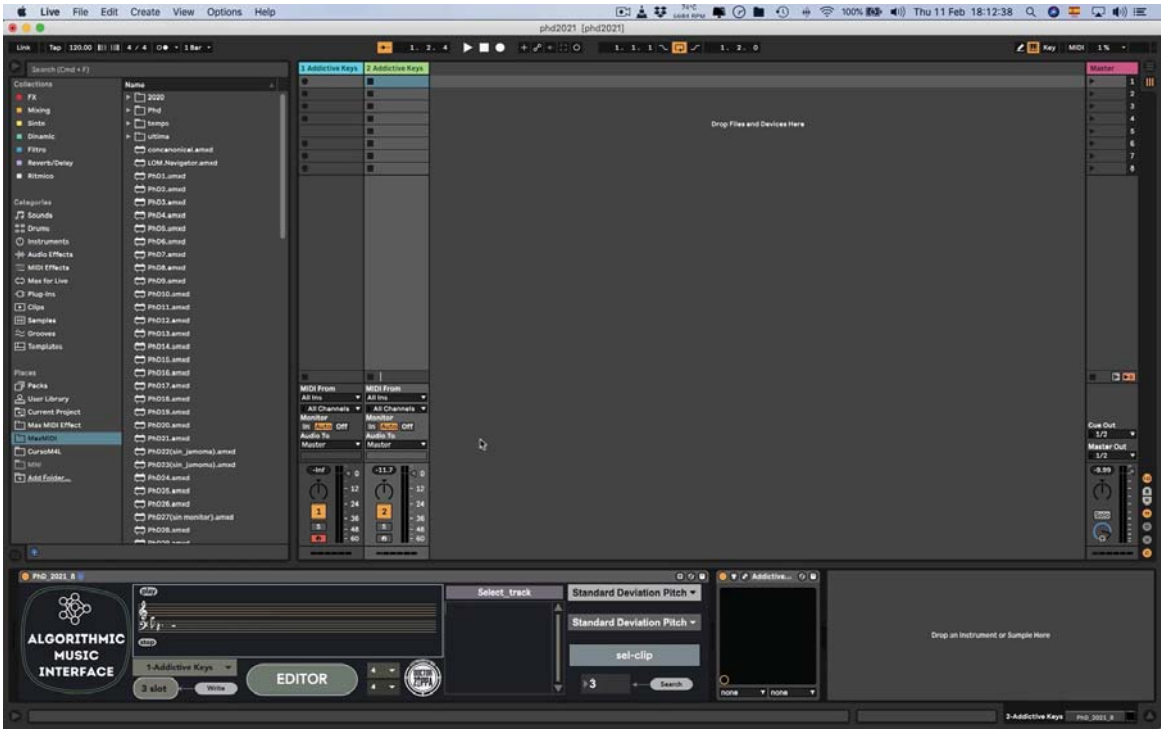
Figura 183: dinámicas obtenidas

Si se utiliza cualquiera de lo otros generadores, es posible, ahora sí, especificar el rango dinámico como se ve en el siguiente video:



Video 34: ejemplo de perfil dinámico caótico

3.5 - Interacción entre los módulos generadores y *Max for Live*



Video 35: editor

Todo lo explicado en este capítulo hace referencia a la interfaz desde donde se generan alturas, ritmos y dinámicas y a la que se accede desde la ventana principal (figura 184) cuando se acciona el botón “EDITOR”. A continuación se explicará cómo la herramienta *AMI* implementa el sistema por el cuál estos datos generados conforman una partitura y cómo se puede volcar dicha partitura a un clip de *Max for Live*.

3.5.1 - Partitura de la ventana principal

Una vez que el usuario genera una serie de alturas a partir de cualquiera de los algoritmos disponibles, esas alturas se escriben, cuando presinonaos el botón “pitch” en la partitura que se ve en la interfaz de la ventana principal y que se denomina *partitura master*. Como todavía no se le adjudicaron duraciones a cada nota, estas se registran utilizando corcheas. Cuando se especifica un ritmo a través de la interfaz dedicada a tal fin, y se presiona el botón “rhythm”, la *partitura master* se modifica incorporando los ritmos generados. Lo mimo sucede con las dinámicas cuando se presiona el botón “dyn”. Así al generar y escribir cada parte por separado, se puede modificar una sin afectar a la otra. Es decir que es posible generar un nuevo perfil de alturas y al presionar el botón “pitch” las notas de la *partitura master* se modificarán pero dejando el ritmo y la dinámica intactos. Lo mismo se puede hacer con los otros dos parámetros. Esto permite al usuario escuchar el resultado al presionar el botón ”play” dentro de la *partitura master* y, si por ejemplo las alturas y el ritmo son de su agrado pero la distribución dinámica no lo es, puede cambiarla sin modificar los otros dos parámetros. Una vez hecho esto es posible escribir los datos almacenados en la *partitura master* en uno de los clips de Live de cualquiera de los tracks que se encuentren en la sesión.

Figura 184: Interfaz de usuario

Para ello debe escoger uno de los tracks existentes en el menú que se ve en la figura 184 (1) y el número de slot donde lo escribirá. Luego simplemente presionando el botón “Write” la secuencia se escribe en dicho clip. Hay que aclarar que el nombre de los tracks existentes se actualizan automáticamente, es decir que si el usuario borra, agrega o cambia el nombre de un track, estos cambios se reflejarán en el menú (1). Una vez que esos clips son registrados es posible manipularlos con todas las herramientas y posibilidades que brinda *Ableton Live*.

Capítulo 4. Análisis y clasificación de los parámetros generados

4.1 - Introducción

La herramienta *AMI* implementa, además de los generadores ya expuestos, un sistema desde donde se pueden clasificar los datos obtenidos (figura 185).

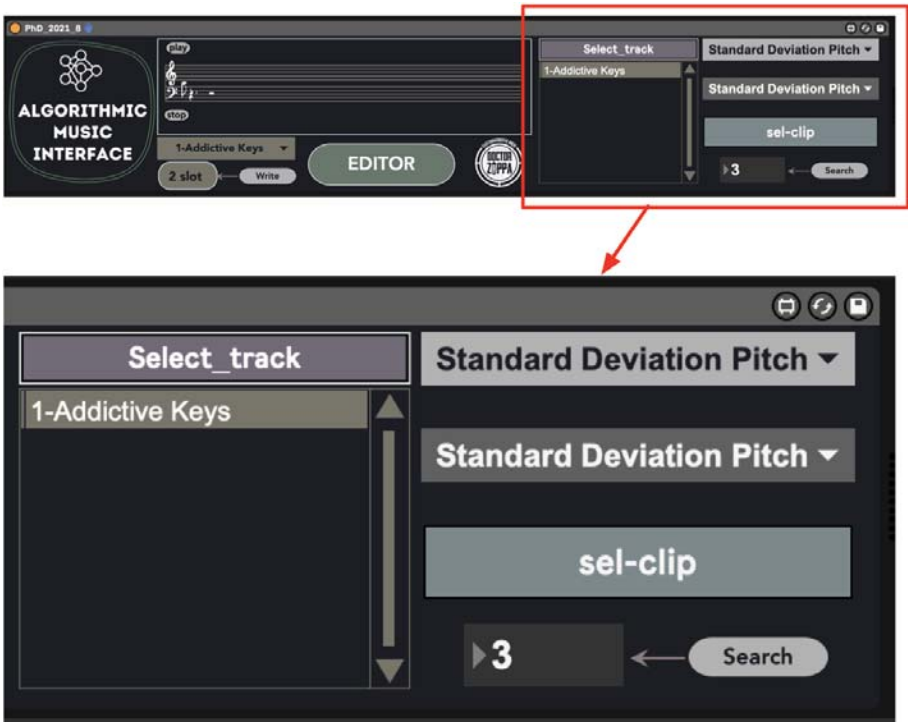


Figura 185: interfaz para la clasificación de los datos generados

Este sistema de clasificación, que será expuesto en el siguiente capítulo, utiliza para cumplir su tarea una serie de técnicas de análisis que incluyen algunas de las que se vieron hasta aquí (como la similitud cronotónica y la relación IcVSIM) y otras que serán explicadas a continuación. Estas técnicas de análisis musical por computadora pueden incluirse, junto con muchas otras que han sido investigadas desde la década de 1960 (Kassler, 1966), en una disciplina llamada *recuperación de datos musicales* o MIR, por su nombre en inglés (Music Information Retrieval).

4.2 - Music Information Retrieval

Stephen Downey define a MIR como “un esfuerzo de investigación multidisciplinar que se empeña en desarrollar esquemas de búsqueda innovadoras basadas en los contenidos, en interfaces novedosas y en mecanismos de entrega en evolución, en un esfuerzo por hacer accesible a todos el vasto catálogo de música del mundo” (Downie, 2004). Estas búsquedas, de las que hacen uso entre otras disciplinas, las ciencias computacionales, las ciencias cognitivas y la musicología (Neubarth, Bergeron, & Conklin, 2011) utilizan estas herramientas para, como su nombre lo indica, recuperar datos subyacentes dentro de un corpus musical dado. En el caso de la musicología, de hecho, existe un área específica a la que se la denomina *musicología por computadora* (Volk, Wiering, & Kranenburg, 2011).

Es posible distinguir dentro de MIR dos grandes ramas: la recuperación de información a partir de datos simbólicos (partituras, MIDI, MusicXML, etc) y la recuperación de información a partir de un archivo de audio (H. Li et al., 2017). Esta tesis se ocupa de los sistemas de recuperación simbólica.

Como ya se mencionó, Michael Kassler propuso, en 1966, un sistema de recuperación de datos simbólicos y un lenguaje de programación creado para tal fin denominado MIR. En su artículo dice:

> El hecho de que *MIR* esté diseñado para su uso en la recuperación de información musical refleja que cualquier función proposicional teórico-musical, efectivamente computable, es decir, cualquier predicado cuyo valor de verdad sea computable a partir de notas, silencios, claves y otros "símbolos primitivos" de notación musical que constituyen una u otra composición particular, se pueden representar como un programa en MIR. (Kassler, 1966).

Mucho se ha avanzado desde aquellos años, no solo por el obvio hecho de que la capacidad de las computadoras ha crecido exponencialmente y porque el acceso a las mismas se ha vuelto infinitamente más común desde la década de 1960, sino también porque las técnicas desarrolladas y los programas (software) creados para tal fin también han crecido de manera importante. Como muestra del desarrollo y el interés que ha suscitado esta disciplina baste mencionar la existencia de la “International Society of Music Information Retrieval” ISMIR[99] que viene realizando sus conferencias internacionales anuales ininterrumpidamente desde el año 2000. En su web esta organización destaca que :

> La Sociedad Internacional de Recuperación de Información Musical es una organización sin fines de lucro que busca promover el acceso, la organización y la comprensión de la información musical. Como campo, la recuperación de información musical se centra en la investigación y el desarrollo de sistemas computacionales para ayudar a los humanos a comprender mejor estos datos, extrayendo de un conjunto diverso de disciplinas, que incluyen, entre otras, la teoría musical, la informática, la psicología, la neurociencia, la bibliotecnología, la ingeniería eléctrica y el aprendizaje automático.

4.3 - Similitud melódica, rítmica y de dinámica

Si bien el campo de la recuperación de datos simbólicos a través de la computadora se encarga de investigar temas tan diferentes como medir la similitud de estilo musical de dos canciones a partir de archivos MIDI (Ens & Pasquier, 2020) o la de buscar la manera de predecir estructuras musicales contenidas dentro de un archivo de datos simbólicos (Carvalho & Bernardes), el presente trabajo se enfoca en los métodos desarrollados para la búsqueda de similitudes y relaciones a nivel melódico, rítmico y dinámico entre dos secuencias de datos (en este caso MIDI) dadas. Algunas de las técnicas que se

[99] https://www.ismir.net/conferences/

utilizan aquí tienen como meta lograr una comparación precisa, lo más objetiva posible, como pueden ser la medida geométrica de similitud o las que implementan sistemas de *n*-gramos, mientras que otras ofrecen una comparación más imprecisa, relacional y por lo tanto mas subjetiva como puede ser la cantidad de entropía que poseen diferentes secuencias o el valor promedio de una serie de alturas. Esto se debe a que la herramienta *AMI* no pretende detectar estas similitudes con un fin analítico sino aportar pistas que el compositor pueda utilizar de manera creativa.

La búsqueda de dichas similitudes, es decir, encontrar semejanzas entre dos frases musicales dadas, implica una tarea compleja ya que involucra diferentes disciplinas en el proceso: la teoría musical, la etnomusicología, la ciencia cognitiva y la informática, todas las cuales deben considerarse simultáneamente (Velardo et al., 2016). Sin embargo, el presente trabajo se decanta por los mecanismos que implementan una comparación a partir de procesos matemáticos, sin considerar otros factores cognitivos y culturales.

4.3.1 - Implementación en *AMI*

Diferentes autores coinciden en que las técnicas más efectivas y por lo tanto las mas utilizadas en el campo de la recuperación de datos musicales son: el uso la distancia de edición, la implementación de sistemas que utilizan *n*-gramos, las que utilizan comparaciones geométricas, el uso de la métrica de Wasserstein, también denominada Earth mover's distance y las que utilizan cadenas de Markov (Downie, 1999; Hoos, Renz, & Görg, 2001; Müllensiefen & Frieler, 2006; Müllensiefen & Pendzich, 2009; Schuitemaker, 2020; Typke, Wiering, & Veltkamp, 2007; Urbano, Lloréns, Morato, & Sánchez-Cuadrado, 2010). Todas estas técnicas han sido estudiadas y utilizadas para la búsqueda de

similitudes melódicas precisas. Como ya se mencionó, este trabajo no se propone crear una herramienta para la búsqueda de similitudes que posean un alto nivel de precisión, como las que se utilizarían, por ejemplo, para la detección de plagios, sino que busca brindarle al compositor algunas pistas de posibles relaciones entre las secuencias generadas a partir de los procesos expuestos en los capítulos 3 y 4 y que el usuario puede utilizar para "determinar la naturaleza de una identidad musical y para resolver mejor el problema relacionado con el principio de integración" (Baboni-Schilingi & Voisin, 1999), es decir como una sugerencia para el encadenamiento de diferentes secuencias. Por lo tanto se han incorporado otras medidas de comparación menos exactas como las obtenidas a partir de ciertos análisis estadísticos (desviación estándar, promedio), de la teoría de la información (entropía) y de técnicas especificas de comparación de grupos de alturas y ritmos como son las medidas de IcVSIM y cronotónica, ya explicadas en el capítulo 3. Los algoritmos de análisis utilizados en *AMI*, que serán usados luego para la clasificación de los clips almacenados en la sesión de *Ableton Live*, son:

- Para las alturas: desviación estándar, entropía, IcVSIM, distancia de edición y perfil de la distancia de edición.
- Para el ritmo: entropía y similitud cronotónica.
- Para las dinámicas: promedio, entropía y perfil de la distancia de edición.

Alturas: desviación estándar

En el capítulo 3 se explicó que la desviación estándar de una cadena de valores es la medida de dispersión que posee dicha lista con respecto a la media y que se calcula a partir de la raíz cuadrada de la varianza (Di Liscia, 2011; Salazar & Castillo, 2018). Es

posible ver la formula para su cálculo en la ecuación 31 y un ejemplo en la figura 153. Aquí, sin embargo, se utiliza esta medida para comparar la cantidad de desviación que posee cada secuencia y poder clasificarlas luego a partir de dicho valor.

En en el ejemplo de la figura 186 se ven cuatro secuencias donde las alturas son generadas utilizando caos logístico con el mismo valor para *r* y donde el ritmo y la dinámica son siempre las mismas. Se puede observar allí que, aunque las cuatro secuencias son muy similares, el sistema arroja un valor diferente de desviación estándar para cada una.

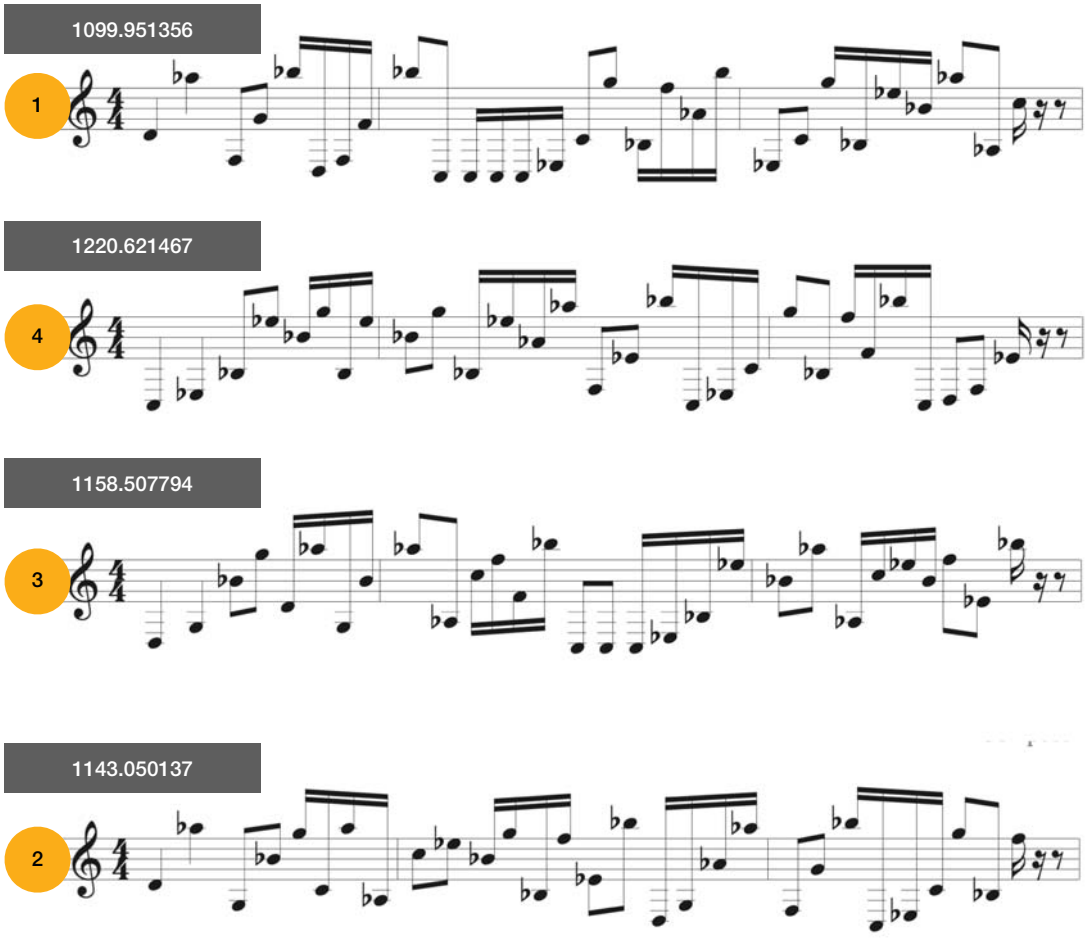
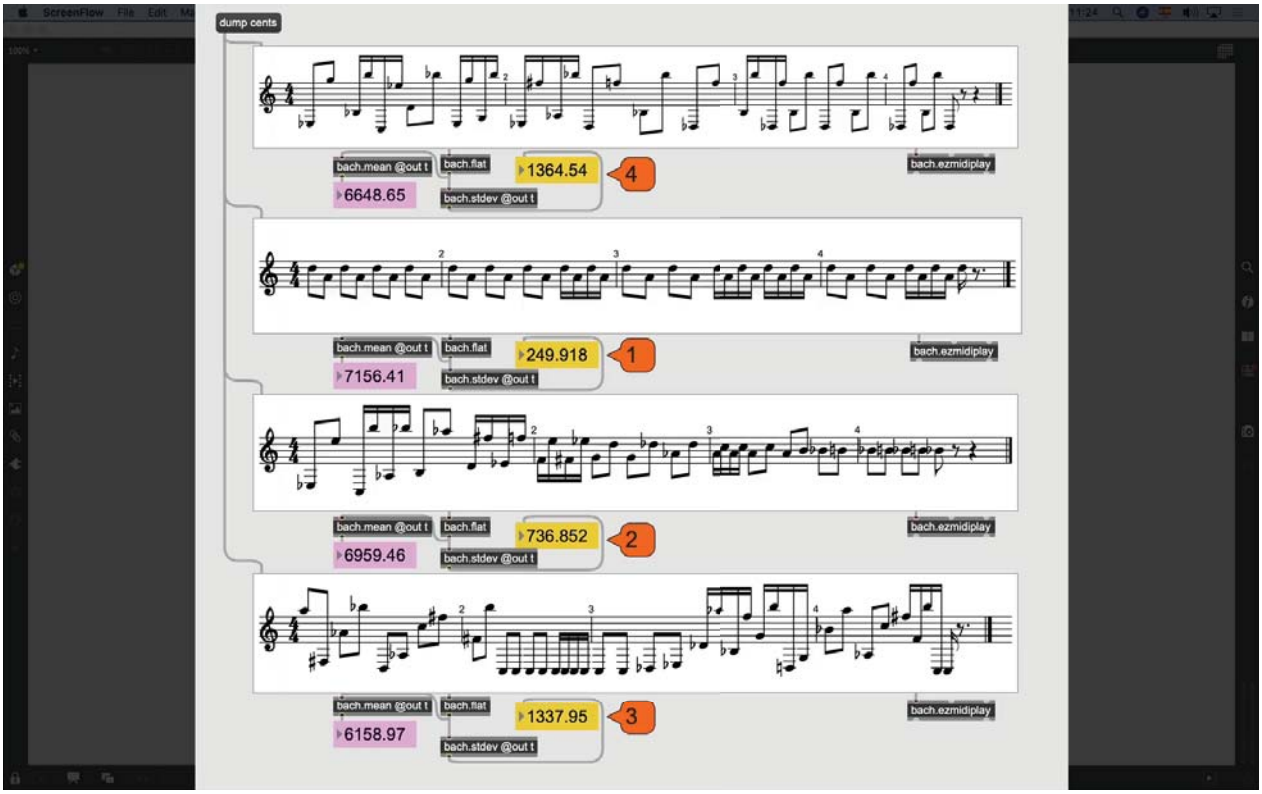


Figura 186: desviación estándar ejemplo 1

El ejemplo del video 36 es más claro. Allí se aprecia que la segunda secuencia posee una desviación de 249.918, la tercera de 736.852, la cuarta de 1337.95 y la primera de 1364.54, es decir que la segunda secuencia posee, como es obvio, una menor desviación con respecto a las demás. Así la clasificación de estas queda (ordenado de menor desviación a mayor) : 1 → 4, 2 → 1, 3 → 2 y 4 → 3.



Video 36: desviación estándar

Es pertinente aclarar aquí, aunque quizá resulte obvio, que este valor, como el de la entropía, no implican una medida de comparación, sino una medida absoluta, es decir que la desviación estándar de una secuencia se calcula con respecto a sí misma. No es así para la relación IcVSIM, por ejemplo.

Alturas: entropía

Se define entropía como una medida del desorden de un sistema. Si bien es una magnitud tomada de la termodinámica, fue Claude E. Shannon, estudiando la teoría de la informacion (Shannon, 1948), quien la introdujo como medida de desorden o incertidumbre de una fuente de informacion.

En la librería *Morphologie* de *OpenMusic* se encuentra implementado un objeto llamado "e-Shannon" que calcula esta medida, esto como ejemplo de que ya ha sido utilizado como método de análisis de secuencias musicales. A propósito de ello, en la documentación de dicha librería se lee: “La entropía mínima (0.0) se alcanza cuando todos los elementos son idénticos. Es máximo cuando cada clase contiene el mismo número de elementos. Si consideramos una secuencia aleatoria como una serie de valores sucesivos con una probabilidad igual para cada elemento, entonces la entropía puede ser un índice del grado de aleatoriedad de la secuencia.”. (Baboni-Schilingi & Voisin, 1999). La fórmula para calcular la entropía presente en una cadena de valores es:

$$H = -K \sum_{i=1}^{n} p_i \log(p_i)$$

Ecuación 35

Donde *K* es una constante y p_i es la probabilidad del estado i donde $0 < i \leq 1$. Para calcular p_i y normalizarlo a 1 se realiza el siguiente cálculo:

$$p_i = \frac{Z_i}{\Gamma}, \ \Gamma = \sum_{i=1}^{n} Z_i$$

Ecuación 36

Donde Z_i es el iésimo valor del histograma resultante de la lista de valores a calcular y Γ es igual a la suma de todos los valores de dicho histograma. En la figura 187 se observa el cálculo de la entropía de las mismas secuencias que se utilizaron en el ejemplo del video 36. Se observa que aquí la secuencia 2 y la 4 se enrocaron, es decir, ahora la clasificación queda: 1 → 2, 2 → 1, 3 → 4 y 4 → 3.

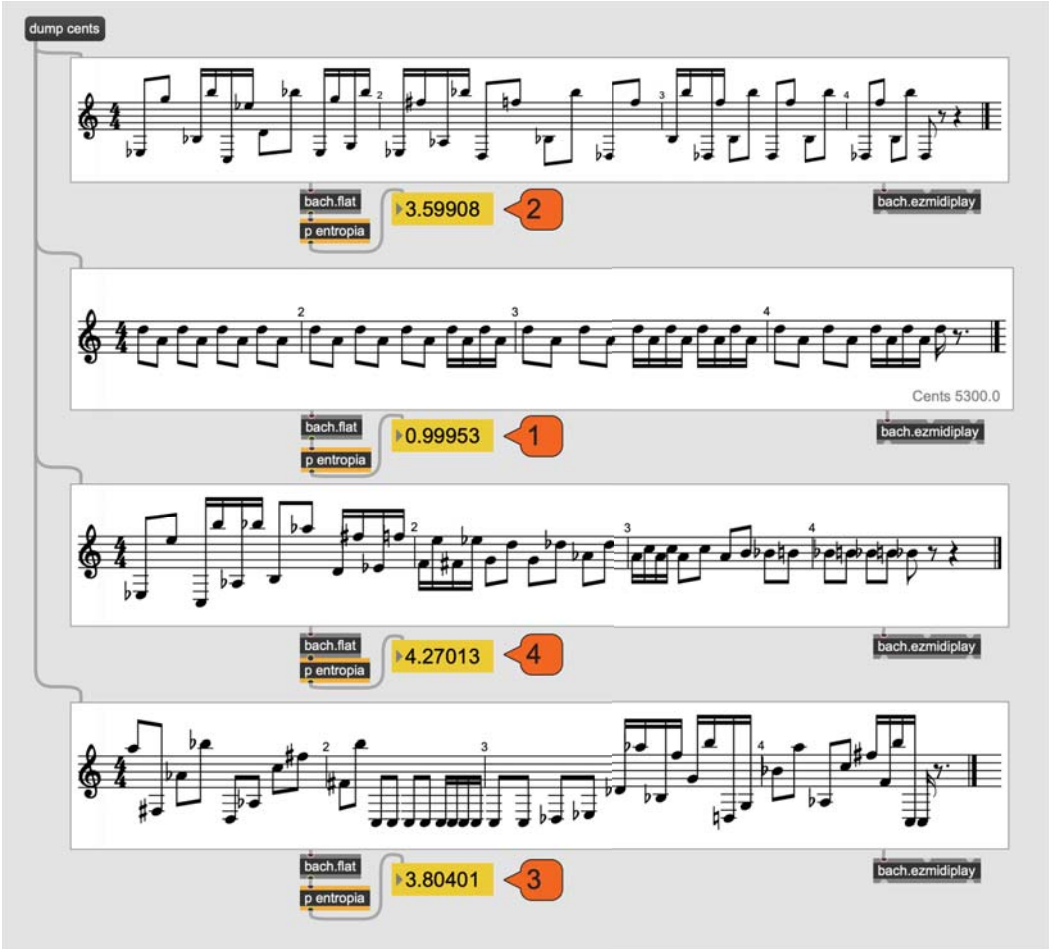


Figura 187: ejemplo de entropía

En la siguiente figura se ve cómo ha sido implementado este cálculo en *Max*:

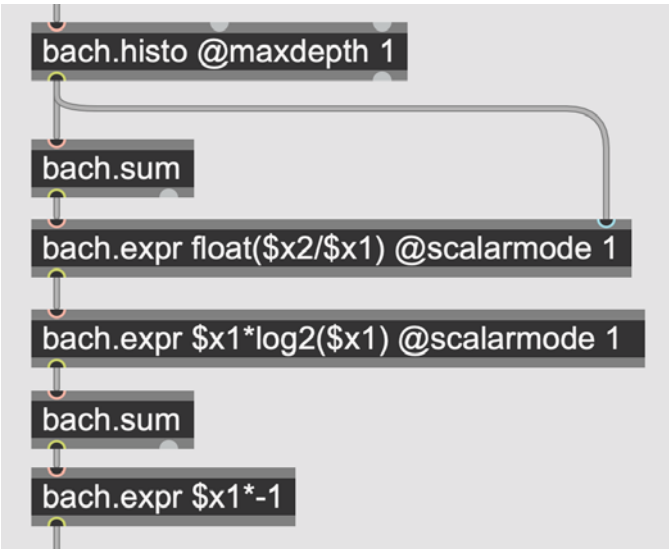


Figura 188: calculo de la entropía

Alturas: distancia de edición

En la distancia de Levenshtein o distancia de edición (Levenshtein, 1966), dos secuencias de símbolos se asemejan más o menos entre ellas de acuerdo a la cantidad de operaciones que hay que realizar para transformar a una en la otra. Por operaciones se refiere a la inserción, eliminación o la sustitución de un símbolo. Cuanto menos operaciones se necesiten para lograr esto, mayor es la similitud entre las cadenas. Puede utilizarse para comparar cadenas de alturas, ritmos o dinámicas. El primer paso es siempre computar el intervalo (de alturas, duraciones, velocity) entre las notas que conforman las secuencias para que sean estas las que serán comparadas y no los valores en sí, ya que esto no serviría para detectar, por ejemplo, la igualdad de dos secuencias exactas pero transportadas (Schuitemaker, 2020). Las tres operaciones mencionadas tienen, cada una, un peso o valor definido:

- Inserción: se denota como $(\lambda \rightarrow \alpha)$ donde λ representa el símbolo vacío (la ausencia de un símbolo). Esta operación tiene un valor de 1.
- Eliminación: se denota como $(\alpha \rightarrow \lambda)$ y también tiene un valor de 1.
- Sustitución: se puede sustituir un símbolo por otro idéntico $(\alpha \rightarrow \alpha)$, y en ese caso el valor asignado será 0 o por uno diferente $\left(\alpha \rightarrow \beta\right)$ y entonces tendrá un valor de 1.

Entonces si se comparan dos secuencias como las de la figura 189, denotando a la superior como *A* y a la inferior como *B* se obtiene:

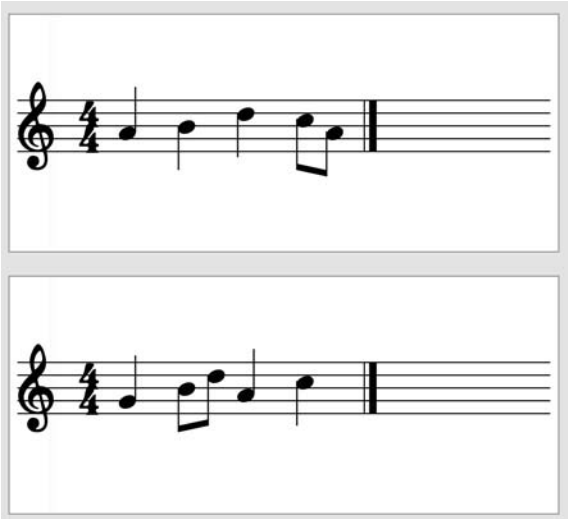


Figura 189: melodías de ejemplo

Sustitución (*B*: Sol → *A*: La) = 1
Sustitución (*B*: Si → *A*: Si) = 0
Sustitución (*B*: Re → *A*: Re) = 0
Sustitución (*B*: La → *A*: Do) = 1
Sustitución (*B*: Do → *A*: La) = 1

Por lo tanto la similitud entre *A* y *B* tendrá un valor de 3. Este cálculo se realiza a partir de una matriz que tiene un tamaño de *m* x *n,* siendo *m* el tamaño de la secuencia *A* y *n* el tamaño de la secuencia *B*. El valor de la distancia de edición será dado por el valor de $d_{n,m}$. Para ello se utiliza el siguiente calculo recursivo:

$$d_{0,0} = 0$$

$$d_{i,j} = min \begin{cases} d_{i-1,j} + 1 \\ d_{i,j-1} + 1 \\ d_{i-1,j-1} + (\text{if } a_i = b_j \text{ then } 0 \text{ else } 1) \end{cases}$$

Ecuación 37

En la figura 190 se puede constatar que, al aplicar el algoritmo de la ecuación 37 se obtiene el resultado deseado.

	λ	La	Si	Re	Do	La
λ	0	1	2	3	4	5
Sol	1	1	2	3	4	5
Si	2	2	1	2	3	4
Re	3	3	2	1	2	3
La	4	3	3	2	2	2
Do	5	4	4	3	2	3

Figura 190: ejemplo de distancia de edición

A continuación se muestra su implementación en *Max*. Como ya se mencionó, este cálculo utiliza una recursión. Si bien se podría haber intentado, para esta tarea, utilizar el objeto **gen** que se usó cuando se implementaron los algoritmos de generación (capítulo 3), se ha escogido aquí utilizar *JavaScript* ya que el repositorio http://rosettacode.org proporciona en su base de datos un código muy eficiente para calcular la distancia de Levenshtein y *Max* puede utilizar dicho lenguaje de programación a través de su objeto **js**. A continuación se muestra el código en cuestión en la versión adaptada por Julien Vincenot[100] para el objeto **js**:

```
inlets = 2;
outlets = 1;

var str1;
var str2;

function list()
[
        if(inlet==0)
        [
        str1 = arrayfromargs(arguments);
        outputRes()
        ]
        else if(inlet==1)
        [
        str2 = arrayfromargs(arguments);
        ]
]

function levenshtein(str1, str2) [
  var m = str1.length,
     n = str2.length,
     d = [],
     i, j;

  if (!m) return n;
  if (!n) return m;

  for (i = 0; i <= m; i++) d[i] = [i];
```

[100] https://julienvincenot.com

```
    for (j = 0; j <= n; j++) d[0][j] = j;

    for (j = 1; j <= n; j++) [
        for (i = 1; i <= m; i++) [
            if (str1[i-1] == str2[j-1]) d[i][j] = d[i - 1][j - 1];
            else d[i][j] = Math.min(d[i-1][j], d[i][j-1], d[i-1][j-1]) + 1;
        ]
    ]
    return d[m][n];
]


function outputRes()
[
outlet(0, levenshtein(str1, str2));
]
```

Alturas: distancia de edición del perfil

Como ya se mencionó, otra de las técnicas utilizadas son aquellas que involucran algún tipo de comparación geométrica de una secuencia. Una de ellas es la llamada “Medida geométrica de similitud” (Aloupis et al., 2006; OMaidin, 1998) que consiste en medir las similitudes entre la melodía *A* y *B* midiendo el área que se obtiene de la superposición de las cadenas poligonales (perfil) que resultan de los valores de altura de las notas. En la figura 191 se observa un ejemplo de lo antedicho. Se ven dos melodías, la superior muestra su cadena polígona en rojo y la inferior en azul. En el medio, en gris, se ven las áreas que resultan de superponer una sobre otra. Es claro que aquí también entran en juego las duraciones de las notas, ya que estas determinan el largo horizontal de cada una de las áreas resultantes. Es entonces una técnica que se aplica a la similitud de alturas y duraciones al mismo tiempo.

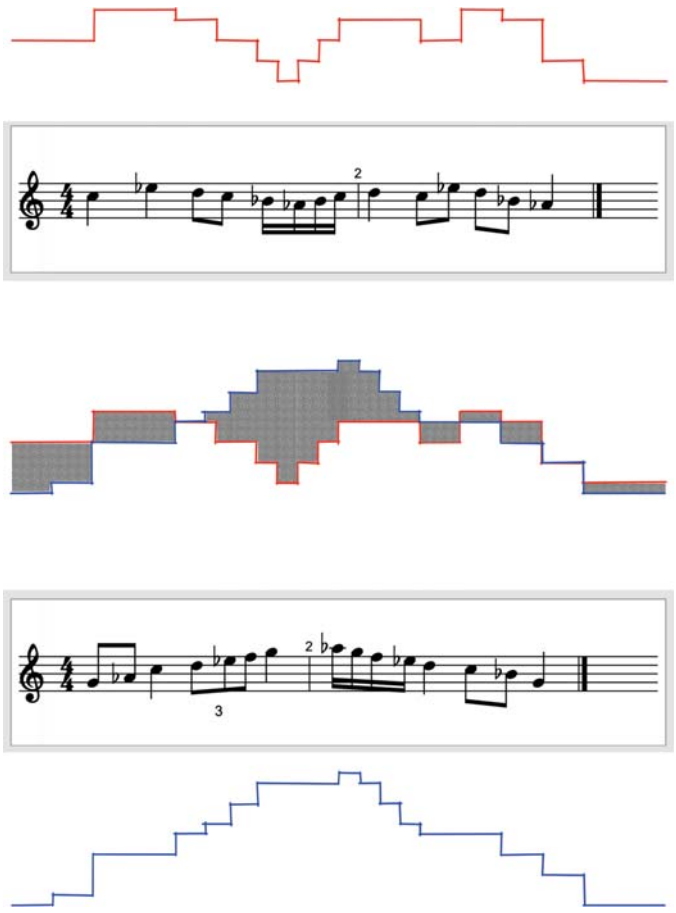


Figura 191: medida geométrica de similitud

En la presente implementación se utiliza una mezcla de dicha técnica con la del cálculo de la distancia de edición. Primero se crea una representación del perfil de alturas de una secuencia, pidiéndole al sistema que si la altura siguiente es mayor coloque un 1, si es menor un -1 y si es la misma un 0:

$$N_0 = 1$$

$$N_{i+1} > N_i = 1$$

$$N_{i+1} < N_i = -1$$

$$N_{i+1} = = N_i = 0$$

Ecuación 38

Por lo tanto la melodía de la figura 192, se traduciría en la siguiente secuencia: [1 1 -1 -1 0 1-1 1]. Como se ve, dos notas iguales en diferente octava no son equivalentes en este caso.

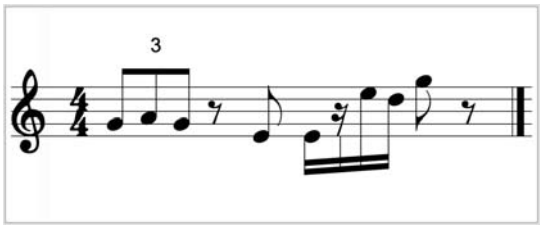
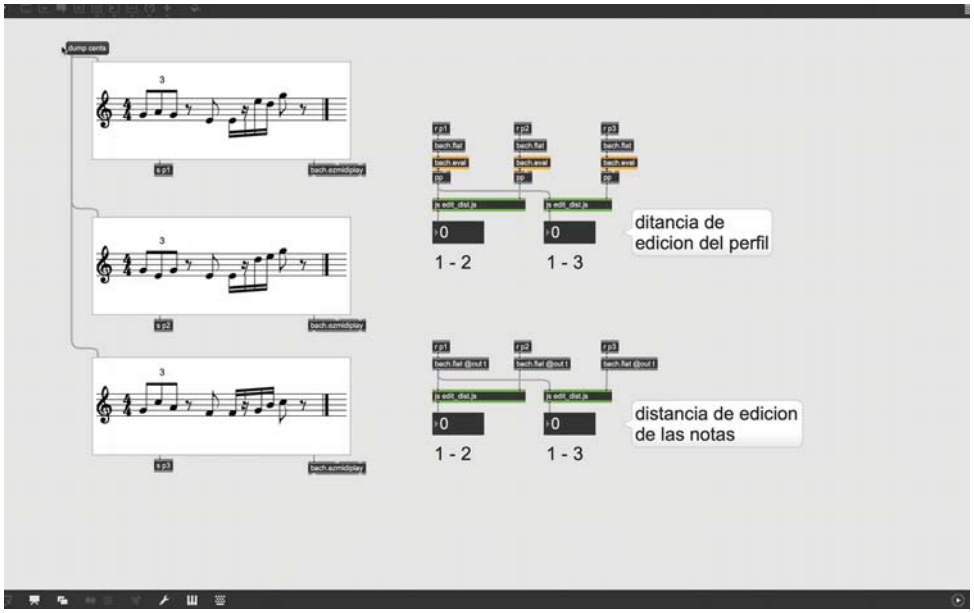


Figura 192: ejemplo de secuencia

Estos perfiles de cada secuencia, representados por los valores 1, -1 y 0, son comparados utilizando la distancia de edición. Así, como se observa en el video 37, la secuencia 1 tiene una distancia de edición del perfil, menor, con respecto a la 3 (distancia de 1), que con respecto a la 2 (distancia de 3), sin embargo esta relación se invierte si se considera la distancia de edición de las alturas (no del perfil). En ese caso la secuencia 1 posee una distancia menor con respecto de la 2 (distancia de 3) que con respecto de la 3 (distancia de 7).



Video 37: distancia de edición

Alturas: n-gramos

Otra técnica es la que utiliza un concepto llamado *n*-gramos (Doraisamy & Rüger, 2002; Downie, 1999; Frieler, 2006; Wolkowicz & Kešelj, 2011). Por *n*-gramos se entiende, en este contexto, secuencias de notas de un largo *n* con superposición de *n-1*. Entonces si se tienen las siguientes notas [Do Re Mib Do Sol] y *n = 2* se obtendrá [Do Re] [Re Mib] [Mib Do] [Do Sol]. Así, entonces, se utilizan estos gramos o términos, para medir la similitud entre secuencias. Daniel Müllensiefen y Marc Pendzich proponen dos estrategias para medir estas similitudes (Müllensiefen & Pendzich, 2009), estas son la medida Ukkonen (ecuación 39) y la medida Sum Common (ecuación 40). La primera cuenta las diferencias de *n*-gramos que hay entre dos cadenas, es decir cuántos de los términos son diferentes, mientras que la otra medida cuenta las similitudes.

$$\sigma(s,t) = 1 - \frac{\sum_{\tau \in s_n \bigcup t_n} \left| f_s(\tau) - f_t(\tau) \right|}{\left| s \right| + \left| t \right| - 2(n-1)}$$

Ecuación 39

$$\sigma(s,t) = \frac{\sum_{\tau \in s_n \bigcap t_n} \left| f_s(\tau) + f_t(\tau) \right|}{\left| s \right| + \left| t \right| - 2(n-1)}$$

Ecuación 40

Donde $f_s(\tau)$ y $f_t(\tau)$ son la frecuencia con que el término τ aparecen en la melodía *t* y *s* respectivamente y s_n y t_n designan el conjunto de términos diferentes en *s* y en *t,* mientras que |s| y |t| designan el tamaño de ambas melodías. El total de *n*-gramos en las dos melodías es, respectivamente, |s| - *n+1* y |t| - *n+1.*

En la herramienta *AMI* se han implementado la medida de Sum Common de dos maneras diferentes. En la primera (que se observa en la figura 193), las mismas alturas pero en diferentes octavas no son equivalentes, mientras que en la segunda implementación sí lo son. En dicha figura se puede observar la programación del algoritmo. Las alturas que ingresan por el primer **inlet** (1) serán las correspondientes al clip seleccionado en la interfaz de *Ableton Live* y por lo tanto serán las que se compararán con el resto de los clips. Dichas alturas son divididas entre 100 (2) para que estas se encuentren dentro de un rango MIDI (0 - 127)[101]. Luego, se crean los gramos propiamente dichos: los valores ingresan a **bach.group** (3) donde los términos son creados (en este caso serán 3 gramos).

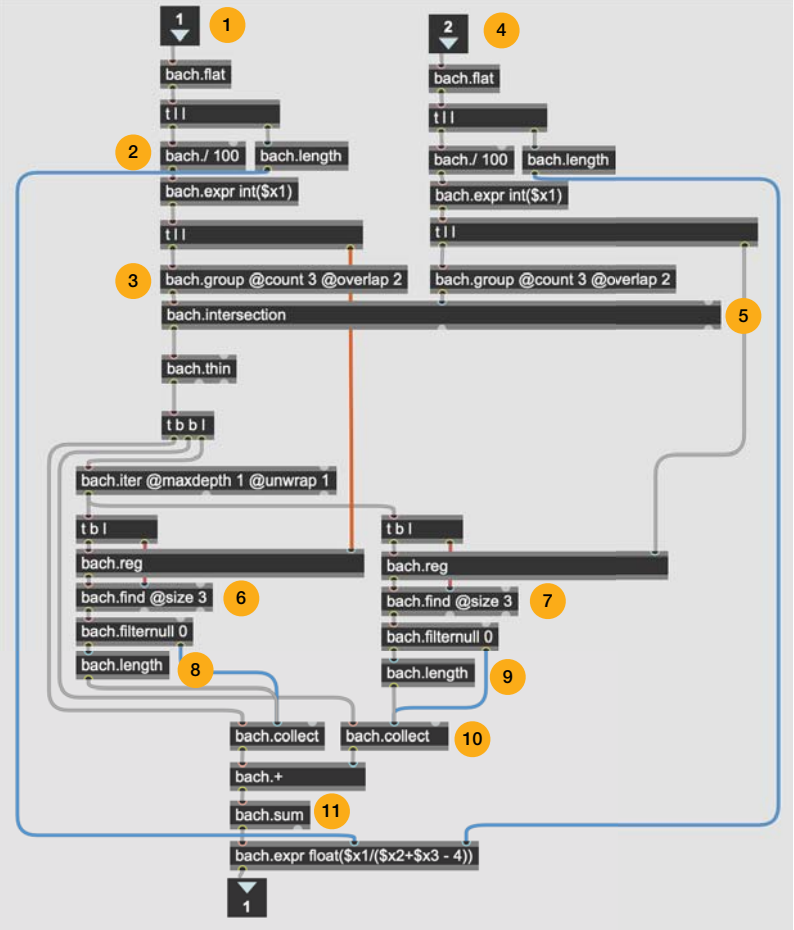


Figura 193: programación para Sum Common

101 Aunque esto no es estrictamente necesario.

Las alturas que se compararán, y que ingresan por el segundo **inlet** (4), atraviesan el mismo proceso. Luego se aplica la parte de la fórmula donde se calcula la intersección de los conjuntos formados por ambos *n*-gramos. A continuación se busca cuántos gramos de la intersección resultante están contenidos en el primer clip y cuántos en los demás clips, esto se implementa con el objeto **bach.find** (6 y 7) y se cuentan esas ocurrencias a través del objeto **bach.length** (8 y 9). Estos valores colectados en sendos **bach.collect** (10) alimentan la implementación del resto de la fórmula en (11).

La otra implementación de Sum Common mencionada *SumCommonEq* utiliza una programación idéntica, con la diferencia que las alturas atraviesan un objeto **bach.mod 12** para que las notas sean equivalentes a pesar de encontrarse en octavas diferentes. Esto hará que sea más probable encontrar coincidencias entre los clips.

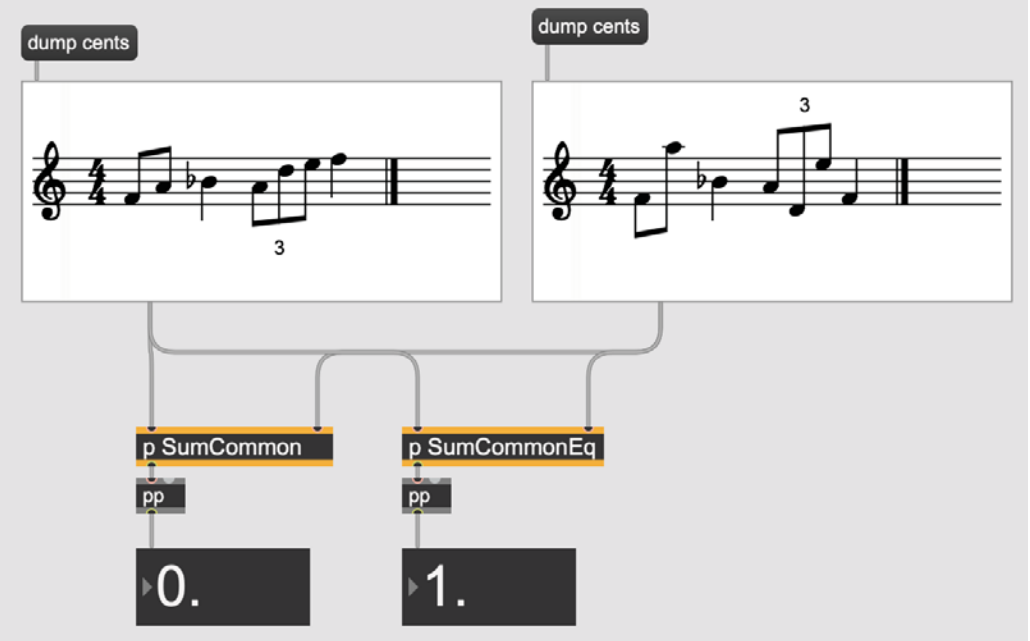


Figura 194: *SumCommon* y *SumCommonEq* compradas

En la figura 194 se aprecia un ejemplo de lo antedicho. La melodía de la izquierda sólo difiere de la derecha en que algunas de sus notas se encuentran en diferentes octavas. Para el algoritmo implementado dentro del objeto **p SumCommon**, entonces, estas dos melodías son totalmente diferentes por lo que arrojan un valor de comparación igual a 0 (el mínimo), mientras que para el algoritmo **p SumCommonEq**, al no distinguir octavas, el valor de comparación es 1 (el máximo).

Ritmo: entropía

Lo dicho acerca de la entropía aplicada a las alturas se aplica aquí a las duraciones. De esta manera este valor indicará cuán desordenado es el grupo de duraciones.

Ritmo: similitud cronotónica

Esta similitud y su algoritmo para calcularla ya ha sido explicada en el capítulo 3. En este caso, la herramienta analiza el nivel de similitud cronotónica que existe entre el clip seleccionado y los demás clips que se encuentran en el mismo track u otros tracks de la sesión.

Dinámica: promedio, entropía y distancia de edición del perfil

En el caso de la dinámica se analizan estos tres datos, el promedio de las velocities, que brinda una idea de la dinámica general de la secuencia, la entropía que informa de cuán pareja o estable es la dinámica y la distancia de edición del perfil de dinámicas, similar al usado con las alturas.

4.4 - Clasificación

A continuación se explicará el sistema de clasificación programado dentro de la herramienta *AMI*. Aquí el objetivo es implementar un sistema que permita clasificar los clips almacenados en cualquiera de los tracks de *Max for Live*, de manera tal que el sistema sugiera una posible sucesión de clips a partir de los diferentes criterios de análisis explicados en el capítulo 4.

4.4.1 - Escritura de clips

Una vez que una secuencia de notas ha sido generada a través de la interfaz de edición que se vio en el capítulo 3, dicha secuencia puede almacenarse en forma de clip en *Max for Live*. Un clip (figura 195) es una secuencia de información MIDI o de audio que se almacena en un casillero (clip slot) dentro de un track. Cada track puede contener miles de clips, por lo tanto este recurso es casi inagotable. Los clips en *Ableton Live* se muestran así:

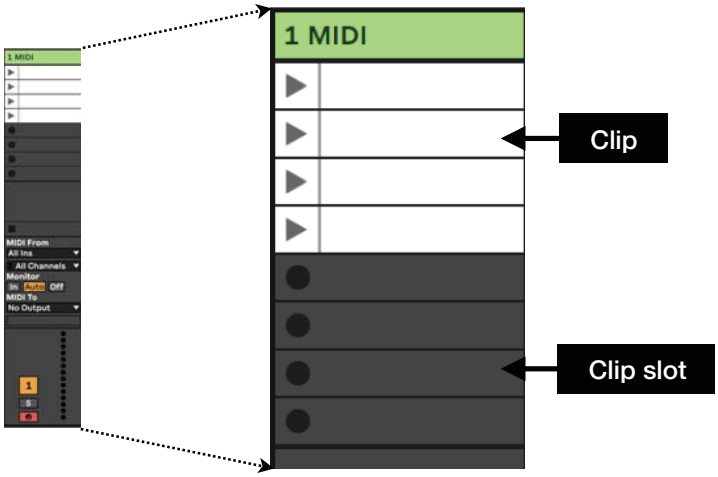


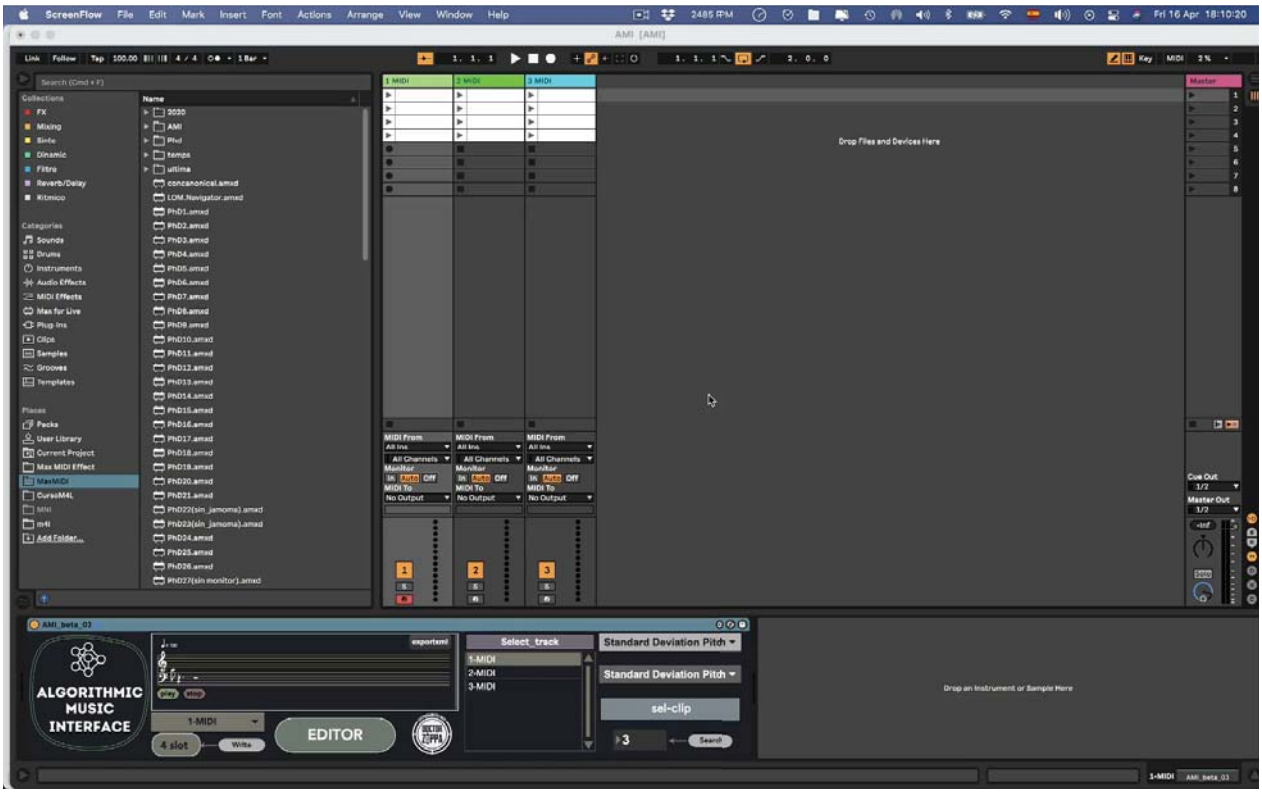
Figura 195: clips en un track MIDI

Para almacenar, entonces, un clip desde *AMI* lo que se debe hacer es, una vez generada y almacenada una secuencia en la partitura master (figura 196 (1)), escoger el track donde se almacenará (2), luego el número de slot (3) y por último hacer clic en el botón “write”.

Figura 196: interfaz *AMI*

4.4.2 - Interfaz para clasificación

En la figura 196 se puede ver también la parte de la interfaz que pertenece a la clasificación. En (5) es posible accionar el botón “Select_track”, al hacer esto es posible escoger los tracks que se desean que participen en la clasificación. Luego en (6) se escogen los dos criterios de análisis a utilizar. Si se desea utilizar un solo criterio, se deben seleccionar el mismo ítem en ambos menús. En (7) se especifica la cantidad de clips que participarán de la clasificación y por último al accionar el botón “sel-clip” (8) se puede escoger el clip que será comparado contra todos los demás que se encuentran en los tracks seleccionados. En el siguiente video se muestra este procedimiento. Se ve primero cómo se clasifican los primeros ocho clips en orden de similitud con respecto al que se escoge haciendo clic. Luego se clasifican tres clips del segundo track y por último cinco clips de los tracks dos y tres.



Video 38: ejemplo de clasificación

Como puede verse en el video 38, lo que se obtiene con este procedimiento es una clasificación numérica, donde el clip seleccionado será siempre el número 1, y luego a partir del 2 se enlistarán los demás clips en orden creciente desde el más similar, de acuerdo a los criterios de análisis escogidos, al menos similar.

Cuando se escoge clasificar los clips de un solo track, esta clasificación puede utilizarse como una sugerencia para escoger el orden de los clips cuando se tocan directamente en el modo de “sesión” de *Live*. En cambio si se clasifican los clips de varios tracks, este sistema no es tan útil porque *Live*, no permite crear una secuencia de clips de diferentes tracks desde el modo de sesión. En ese caso el usuario puede utilizar el modo “arreglo”, donde puede copiar y pegar los clips escogidos en cualquier track que desee.

4.4.3 Implementación en *Max*

Lo que será necesario, en principio, es crear dos bases de datos, una que contenga las alturas, duraciones y velocities de cada clip generado y otra que almacene los resultados de los análisis realizados sobre cada clip. Cuando una secuencia es escrita en la *partitura master*, las alturas, duraciones y velocities de dicha secuencia son enviadas a un objeto **bach.join 4 @triggers 4** que junta estos tres parámetros más el número de identificación del clip donde estos datos fueron almacenados en forma de secuencia MIDI. El primer **4** que acompaña a **bach.join** indica, entonces, que esta lista estará formada por cuatro miembros, mientras que **@triggers 4** especifica que dicha lista debe ser enviada por la salida del objeto cuando este recibe un dato a través del último **inlet**, y como dicho dato es el numero de identificación del clip que se escribe cuando el usuario presiona el botón "write" que se ve en la figura 196 (4), entonces es en ese momento que la informacion completa, es decir, la lista [alturas, ritmo, dinámicas, id], sale a través de **bach.join** para ingresar al objeto **dada.base** donde se almacenará. En la figura 197 se aprecia esta parte de la programación:

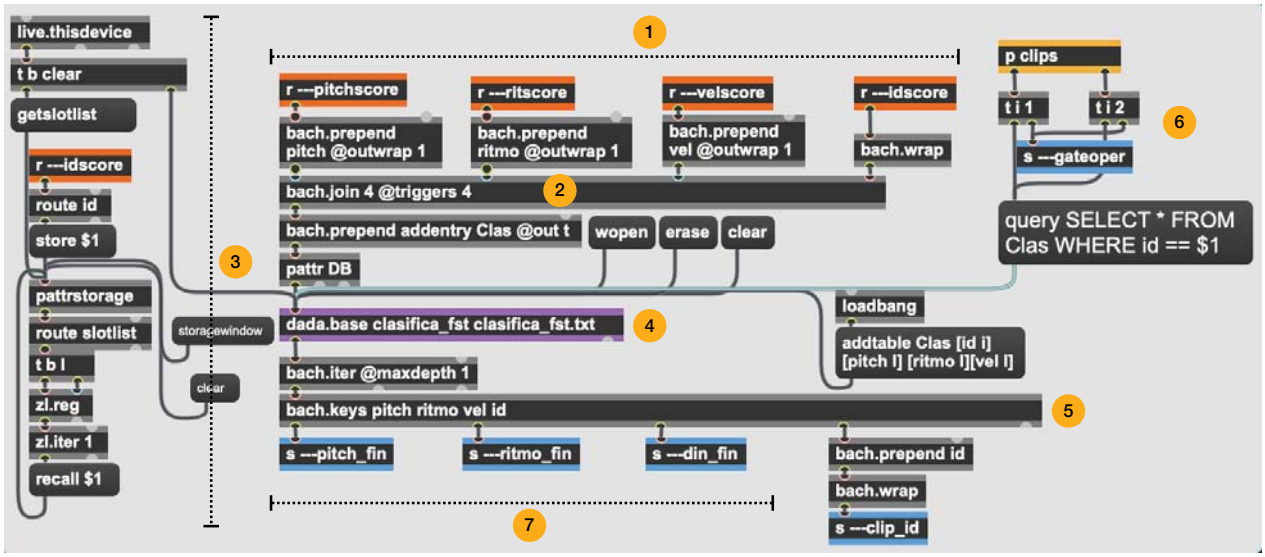


Figura 197: programación para almacenar las secuencias registradas

Estos datos, entonces, llegan a través de sendos objetos **receive**: **r —pitchscore**, **r —ritscore**, **r —velscore** y **r —idscore** al sector que se observa en la figura 196 (1) y, luego de atravesar sendos objetos **bach.prepend** que se encargarán de anteponer a cada información su correspondiente etiqueta, ingresan a **bach.join** donde, como ya se mencionó, se juntarán en una lista única. A continuación dicha lista atraviesa otro **bach.prepend** para agregar, al frente de la misma, el mensaje **addentry Clas** necesario para que **dada.base** (4) la almacene en su interior. Pero antes de ingresar a **dada.base** atraviesa un objeto **pattr DB** que permitirá, en combinación con la programación que se observa a la izquierda de la linea punteada identificada como (3), guardar una memoria de todo lo almacenado incluso cuando se cierre la sesión de *Ableton Live*. Esto permite que el usuario no pierda los datos almacenados cuando cierra su sesión.

Luego, cuando se hace clic sobre un clip determinado, el identificador de dicho clip sale a través de la parte de la programación que se observa en (6) y luego salen todos los identificadores de los demás clips, esto para que el sistema sepa cuál es el clip contra el que se comprarán los demás. Por último el mensaje **query SELECT * FROM Clas WHERE id == $1** le indica a **dada.base clasifica_fst** que envíe por su salida los datos correspondientes a los clips almacenados (primero el escogido y luego los demás) y dichos datos atraviesan **bach.keys** para separarlos por tipo y que sean enviados a través de sendos objetos **send**: **s —pitch_fin**, **s —ritmo_fin**, **s —din_fin** y **s —clip_id**.

A continuación los parámetros provenientes de **dada.base clasifica_fst** atraviesan diferentes módulos donde los sistemas de análisis son aplicados:

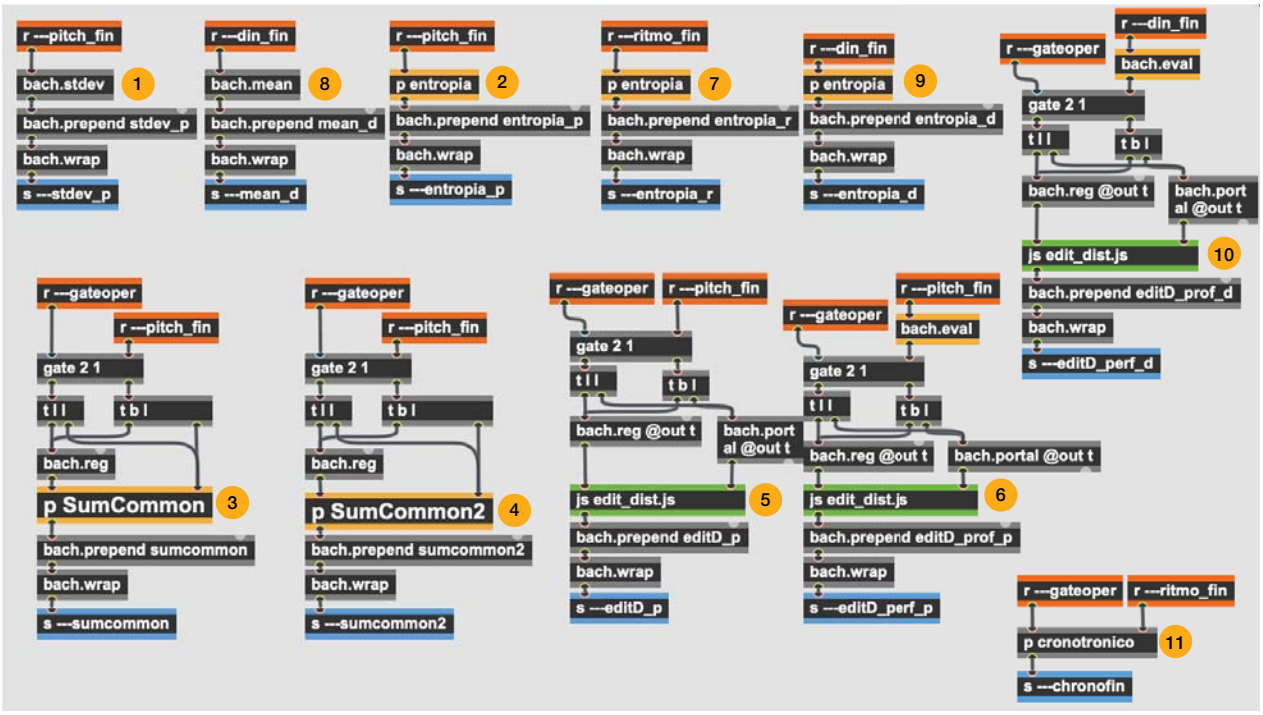


Figura 198: módulos analizadores

Así entonces se obtiene, en el caso de las alturas, el análisis de la desviación estándar (1), la entropía (2), el análisis de tipo *n*-gramos sin equivalencia de octavas (3), con equivalencia de octavas (4), la distancia de edición (5) y la distancia de edición del perfil (6). En el caso de la rítmica, el análisis de la entropía (7) y la distancia cronotónica (11). Por último, para la dinámica, el promedio (8), la entropía (9) y la distancia de edición (10). Es importante destacar que este tipo de programación modular permite, en el futuro, agregar otros tipos de análisis de manera sencilla, ya sea creados por el autor de esta herramienta o por otras personas.

Los datos provenientes de todos estos analizadores, entonces, ingresan a un segundo objeto **dada.base** donde crean una nueva base de datos. En este caso, dicha base de datos tendrá las siguientes entradas:

```
[
        [ Clasdos
                [ stdev_p f ]
                [ entropia_p f ]
                [ sumcommon f ]
                [ editD_p i ]
                [ editD_prof_p i ]
                [ entropia_r f ]
                [ chronotonic f ]
                [ mean_d f ]
                [ entropia_d f ]
                [ editD_prof_d i ]
                [ sumcommon2 f ]
                [ id i ]
        ]
        [ ]
]
```

Es decir que acumulará los valores resultantes del análisis de la desviación estándar de las alturas, la entropía , etc. y por último el identificador del clip analizado. A partir de estos datos, entonces, es posible realizar la búsqueda de los clips que poseen algún tipo de similitud con el seleccionado. Para ello se utiliza aquí el algoritmo del *vecino más próximo*.

> La búsqueda del vecino más próximo es una técnica utilizada principalmente en reconocimiento de formas. Sin embargo, esto no quiere decir que no pueda ser de utilidad en otras disciplinas. Por ejemplo, se puede aplicar para realizar consultas a una base de datos. Estas consultas pueden ser de datos concretos, o incluso de datos aproximados (como por ejemplo la búsqueda de los apellidos parecidos a González (Gonsález, Gosálvez...)). Esta técnica también puede aplicarse a la detección y corrección de errores tipográficos en los procesadores de texto. El método consiste en la búsqueda, en un diccionario de la palabra que más “se parece”, o las *k* que más se parecen, a aquella que queremos corregir. (Andrés, 1996)

Si se coloca en un plano cartesiano las secuencias generadas, donde los valores de sus coordenadas dependen del tipo de análisis seleccionado, se podrá entonces buscar cuáles de los clips se encuentran más cerca del escogido como elemento de comparación. Por ejemplo, si la base de datos posee las entradas que se observa en la figura 200, y se escoge el índice cronotónico para el eje *x* y la entropía de las alturas para el eje *y,* estaríamos creando un plano como el que se ve en la figura 199. Aquí, si el clip seleccionado es el que posee el id 2, el más cercano será el id 5, luego el id 4 y el id 3 (ya que son idénticos) y por último el id 1.

Para implementar esto en *Max* se utilizó, en este trabajo, el objeto de la librería Dada llamado **dada.cartesian** el cual permite, utilizando los datos almacenados en **dada.base**, encontrar al vecino más cercano cuando se le indica cuáles de los datos se utilizarán.

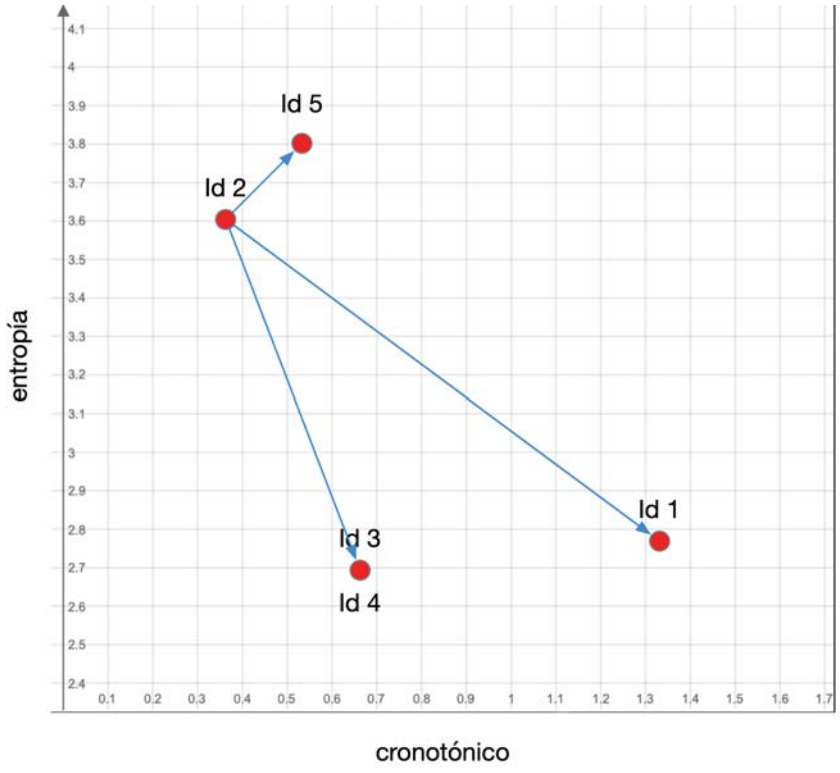


Figura 199: plano cartesiano a partir de la base de datos de la figura 200

```
[
        [
                [ Clasdo_id 1 ]
                [ stdev_p 1362.904491 ]
                [ entropia_p 2.771902 ]
                [ sumcommon 1. ]
                [ editD_p 0 ]
                [ editD_prof_p 0 ]
                [ entropia_r 2.624896 ]
                [ chronotonic 1.324764 ]
                [ mean_d 105.529412 ]
                [ entropia_d 3.499228 ]
                [ editD_prof_d 0 ]
                [ sumcommon2 1. ]
                [ id 86 ]
        ]
        [
                [ Clasdo_id 2 ]
                [ stdev_p 1221.803081 ]
                [ entropia_p 3.616875 ]
                [ sumcommon 0. ]
                [ editD_p 14 ]
                [ editD_prof_p 7 ]
                [ entropia_r 2.840266 ]
                [ chronotonic 0.369651 ]
                [ mean_d 93.647059 ]
                [ entropia_d 3.690117 ]
                [ editD_prof_d 8 ]
                [ sumcommon2 0.066667 ]
                [ id 87 ]
        ]
        [
                [ Clasdo_id 3 ]
                [ stdev_p 1364.781945 ]
                [ entropia_p 2.69866 ]
                [ sumcommon 0. ]
                [ editD_p 12 ]
                [ editD_prof_p 8 ]
                [ entropia_r 2.42893 ]
                [ chronotonic 0.661236 ]
                [ mean_d 93.647059 ]
                [ entropia_d 3.690117 ]
                [ editD_prof_d 8 ]
                [ sumcommon2 0. ]
                [ id 88 ]
        ]
        [
                [ Clasdo_id 4 ]
                [ stdev_p 1364.781945 ]
                [ entropia_p 2.69866 ]
                [ sumcommon 0. ]
                [ editD_p 12 ]
                [ editD_prof_p 8 ]
                [ entropia_r 2.42893 ]
                [ chronotonic 0.661236 ]
                [ mean_d 93.647059 ]
                [ entropia_d 3.690117 ]
                [ editD_prof_d 8 ]
                [ sumcommon2 0. ]
                [ id 88 ]
        ]
        [
                [ Clasdo_id 5 ]
                [ stdev_p 1188.235294 ]
                [ entropia_p 3.807764 ]
                [ sumcommon 0.133333 ]
                [ editD_p 17 ]
                [ editD_prof_p 9 ]
                [ entropia_r 1.89208 ]
                [ chronotonic 0.532056 ]
                [ mean_d 93.647059 ]
                [ entropia_d 3.690117 ]
                [ editD_prof_d 8 ]
                [ sumcommon2 0.133333 ]
                [ id 89 ]
        ]
]
```

Figura 200: base de datos (ejemplo)

En la figura 201 se observa en la parte superior la sección de la programación que se encarga de la búsqueda del vecino más cercano dentro de la base de datos y debajo se muestra un detalle de cada sector de esta programación.

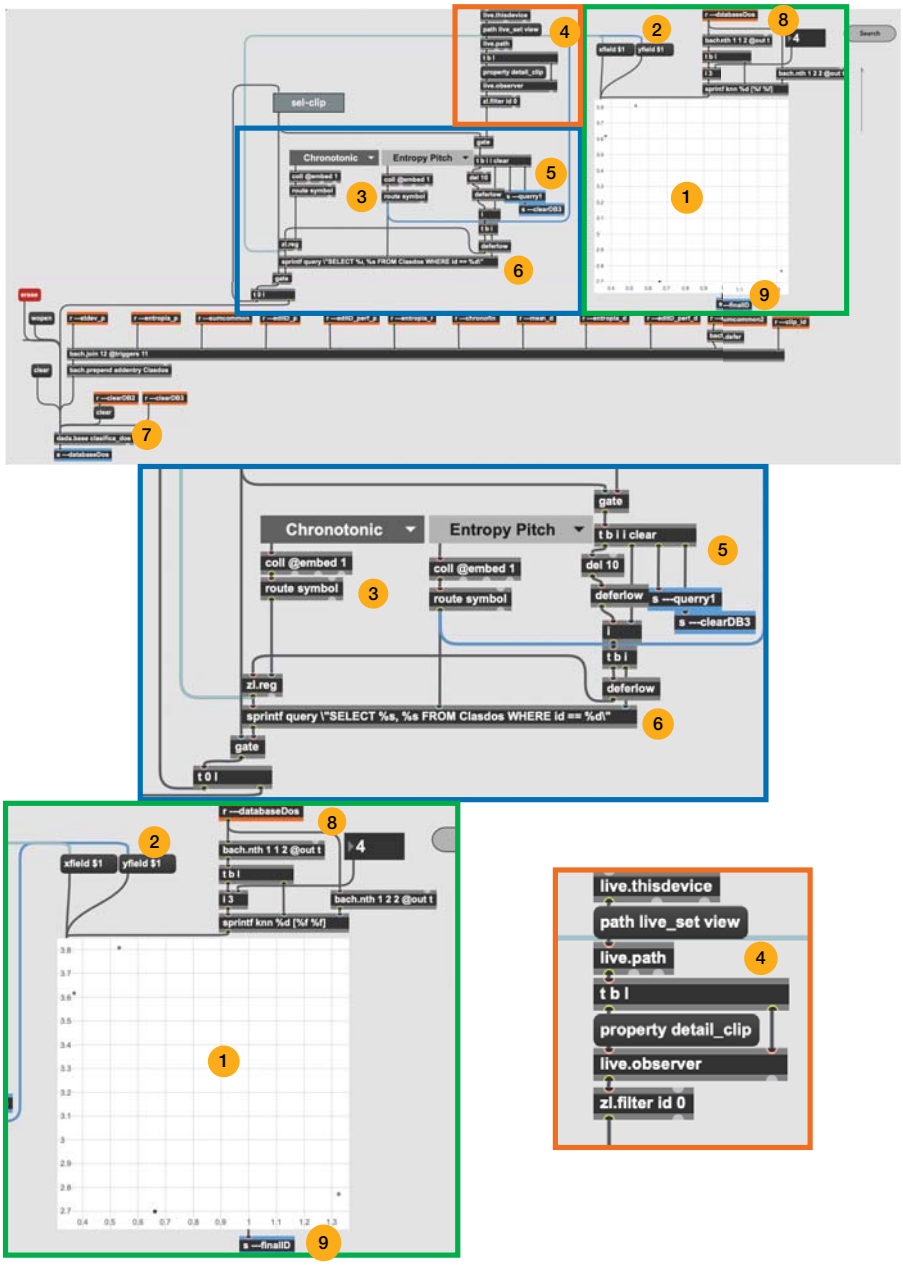


Figura 201: programación de algoritmo de vecino mas cercano

El objeto gráfico **dada.cartesian** es el que se observa en (1). Como puede preciarse, su interfaz es una representación de un plano cartesiano donde los valores del eje *x* e *y* se especifican a través de los mensajes **xfield $1** e **yfield $1** (2). Sin embargo es necesario antes de declarar los ejes, indicarle a **dada.cartesian** el nombre de la base de datos que utilizará y dentro de esta, el nombre específico de la tabla; estas indicaciones se ingresan

desde el *inspector*[102]. Entonces, cuando se especifica un tipo de análisis a través de los menús que se ven en (3), esto le indica a **dada.cartesian** qué datos deben leerse desde la tabla almacenada en **dada.base clasifica_dos**.

En la misma figura, en el sector indicado con un (4) es donde el sistema detecta el identificador del clip que el usuario selecciona en la interfaz de *Ableton Live*. La manera en que este software y *Max* se comunican es a través de tres objetos llamados **live.path**, **live.object** y **live.observer**. El primero se utiliza para establecer la ruta con la que *Max* se comunicará con *Live.* Por ruta se refiere aquí a una dirección que apunta a una función específica dentro de *Live*; estas direcciones cubren casi la totalidad de las funcionalidades de dicho software[103]. Así, a través de **live.path**, en este caso, se accede al set específico donde se está trabajando (esto utilizando el mensaje **path live_set view**), lo que arroja un identificador que ingresa al objeto **live.observer** indicándole a este que debe buscar dentro de dicho set el identificador del clip seleccionado. Esto sucede porque a este último objeto ingresa el mensaje **property detail_clip** que cumple exactamente esa función. En este sentido los objetos **live.observer** y **live.object** son muy similares pero se diferencian en que el primero envía el identificador en cualquier momento que el usuario modifique ese parámetro, mientras que el segundo lo hace cuando el sistema “se lo pregunta”.

A continuación, el identificador del clip seleccionado ingresa al sector indicado con un (5) donde, después de atravesar un **gate** que dejará pasar la información o no dependiendo de si el botón esté encendido o apagado, ingresa a un objeto **t b i i clear** que encausará

---

102 En Max es posible modificar muchas características de los diferentes objetos a partir de la ventana de *inspector* que se abre a través del menú *object -> inspector.*

103 En https://docs.cycling74.com/max8/vignettes/live_object_model?q=LOM se describen pormenorizadamente cada una de estas rutas.

ese identificador a través de cuatro salidas enviándolo secuencialmente de la salida derecha a la izquierda. Así primero se envía el mensaje “clear” a través del objeto **s — clearDB3** a la misma base de datos **dada.base clasifica_dos** para borrar todo su contenido cada vez que se escoge un nuevo clip contra el que se clasificará al resto. Luego sale el número de identificador (recordar que es el id del clip seleccionado) por la segunda salida de **t b i i clear** (contando de derecha a izquierda) y va, a través del objeto **s —querry1** a la primera base de datos, es decir a **dada.base clasifica_fst** para indicarle cuál de los clips almacenados allí será el que servirá como clip de referencia. Esto quiere decir, entonces, que cada vez que el usuario escoge un nuevo clip para clasificar, el proceso de análisis se pone en funcionamiento. Se eligió esta implementación y no una donde los análisis ya estuvieran almacenados, porque hacer eso hubiera implicado almacenar el análisis de todos los clips de la sesión en comparación con sí mismos, lo cual implicaría una base de datos muy extensa y un costo de procesamiento demasiado elevado. Continuando con el objeto **t b i i clear,** luego el id sale por la siguiente salida y este valor se acumula dentro del objeto **i** que lo mantiene allí hasta que recibe un estímulo por su primera entrada y esto sucede cuando, por último, sale el **bang** desde **t b i i clear**. Este estímulo se retrasa 10 milisegundos (utilizando el objeto **del 10**) para darle tiempo al sistema de analizar todos los clips y acumularlos en **dada.base clasifica_dos**. Transcurridos esos milisegundos, y por lo tanto ya completa la base de datos, el id del clip seleccionado ingresa al objeto **sprintf query \"SELECT %s, %s FROM Clasdos WHERE id == %d\”** que reemplazará los símbolos %s, %s y %d por los nombres de los tipos de análisis escogidos y el número de identificador del clip. Así por ejemplo el mensaje resultante podría ser: **query "SELECT entropia_p, editD_prof_p FROM Clasdos WHERE id == 86”**, es decir que se quiere averiguar el valor de entropía de alturas y la distancia de edición del perfil de alturas del clip con el id 86.

El resultado de esta consulta a la base de datos puede ser, por ejemplo, el siguiente: **[[entropia_p 2.638375] [editD_prof_p 0]]**. Este mensaje va, a través del objeto **s — databaseDos** a un sector (8) donde le indica a **dada.cartesian** que debe encontrar *n* vecinos cercanos del punto *x = 2.638375* , *y = 0*. La cantidad de vecinos que deberá buscar se especifica con el número entero que se ve también en (8). Así **dada.cartesian** arroja una lista de id correspondientes a los vecinos del punto escogido, desde el mas cercano al mas lejano, y estos datos son utilizados por el sistema para enumerar los clips en el modo de sesión de *Live*.

Capítulo 5. Casos de estudio

5.1 Primer caso

En este primer caso se construirá un ejemplo basado en la estética de la música electrónica (sin definir un estilo particular) en el cual se utilizará la herramienta *AMI* para generar diferentes partes de las percusiones, el bajo y acordes en un sintetizador.

En el video 39 se muestra paso a paso cómo se construyó este ejemplo y a continuación se explica, en detalle, lo realizado.

Video 39: primer caso que ejemplifica el uso de *AMI*

Primero se generó una secuencia para ser tocada con un sonido de bombo electrónico (aquí se utiliza un emulador de una caja de ritmos Roland 909 [104]). Las alturas no juegan un papel en esta parte ya que el sonido será siempre el del mencionado bombo, por lo tanto solo se generaron 16 notas DO en la octava -2 que es donde este plugin acciona este sonido. Luego se generó un ritmo utilizando el algoritmo de Autómata Celular, donde cada celda es una corchea. El CA inicial en este caso es de 8 celdas:

Figura 202: CA inicial

Por lo tanto el ritmo inicial será:

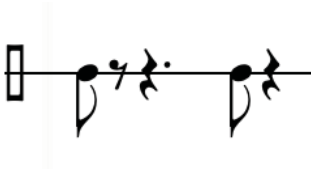


Figura 203: representación musical de la figura 202

Luego la regla escogida es:

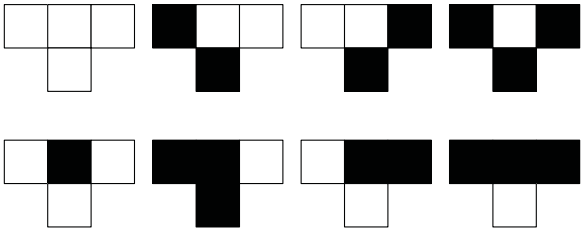


Figura 204: regla utilizada en el primer CA

Como resultado de esto se obtiene lo que se muestra la figura 205. Cabe recordar que este sistema se implementa de forma tal que los casilleros de los extremos se completan repitiéndose, por lo tanto se han agregado dichos casilleros “fantasma” para facilitar la

104 https://www.roland.com/global/promos/roland_tr-909/

comprensión del resultado obtenido.

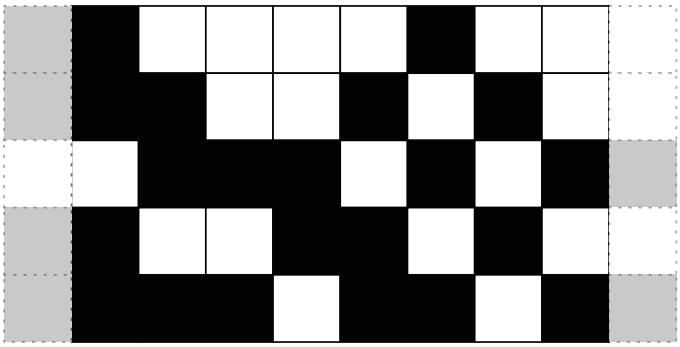


Figura 205: resultado de aplicar al CA original de la figura 203 la regla de la figura 204

Este resultado se traduce en notación musical de la siguiente forma (recordar que se están generado 16 notas):

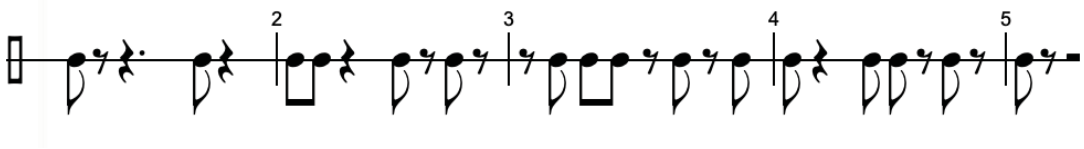


Figura 206: el resultado de la figura 205 en notación musical

En el caso de las dinámicas, también aquí se utiliza, al igual que con las alturas, un solo valor para las velocities. Luego de generar esta secuencia, entonces, es posible escuchar el resultado desde la partitura master, para ello se envía la señal procedente del track denominado “AMI” al track “BD (C1)”. A continuación se almacena el clip 1 en el track mencionado.

A continuación, se generan un grupo de alturas sin modificar el ritmo que ya se había obtenido. La secuencia resultante será interpretada por un sintetizador de bajos de tipo Roland 303[105]. El algoritmo escogidos es el de caos logístico y las alturas con las que será

105 https://en.wikipedia.org/wiki/Roland_TB-303

mapeado son DO, MIb, SOL, LAb y SI dentro de un ámbito de dos octavas (de -2 a -1). Así el resultado es el siguiente:

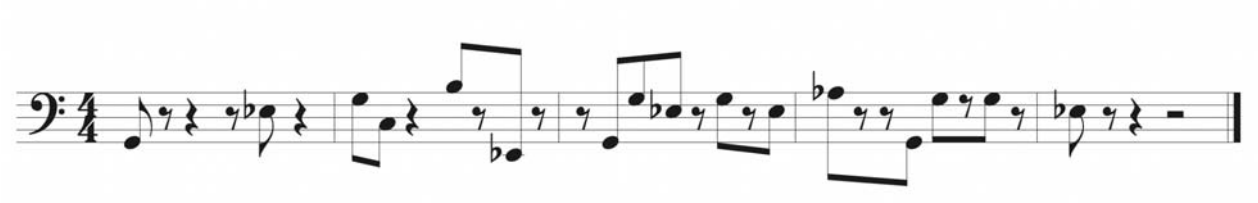


Figura 207: linea de bajo

Esta secuencia resultante se escribe, entonces, en el clip 1 del track “TB 303”. Luego se vuelve a escribir la misma secuencia en el track que contiene un sintetizador analógico virtual tipo Roland Jupiter 8[106] y se traspone una octava hacia arriba. A continuación se generan otras dos secuencias más utilizando la misma rítmica pero generando diferentes cadenas de notas a octavas superiores a la anterior. Así se obtienen acordes que serán almacenados en el mismo track del Jupiter 8. El resultado es:

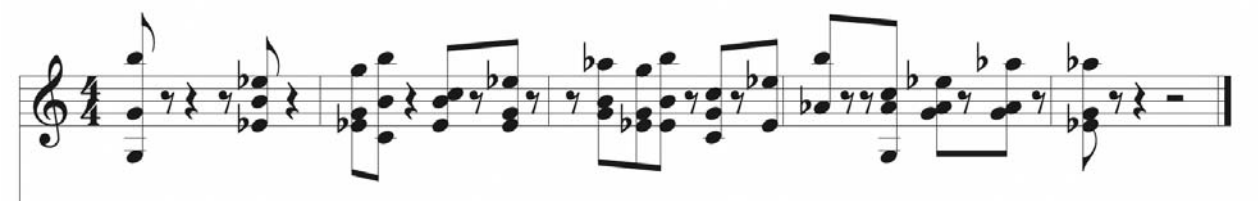


Figura 208: acordes resultantes en el sintetizador

Seguido de esto, el proceso se repite otras tres veces más (usando diferentes algoritmos pero manteniendo las notas utilizadas) obteniendo así cuatro clips en el track de bombo, cuatro en el de bajo y cuatro en el de Júpiter 8. El resultado es el siguiente:

106 https://en.wikipedia.org/wiki/Roland_Jupiter-8

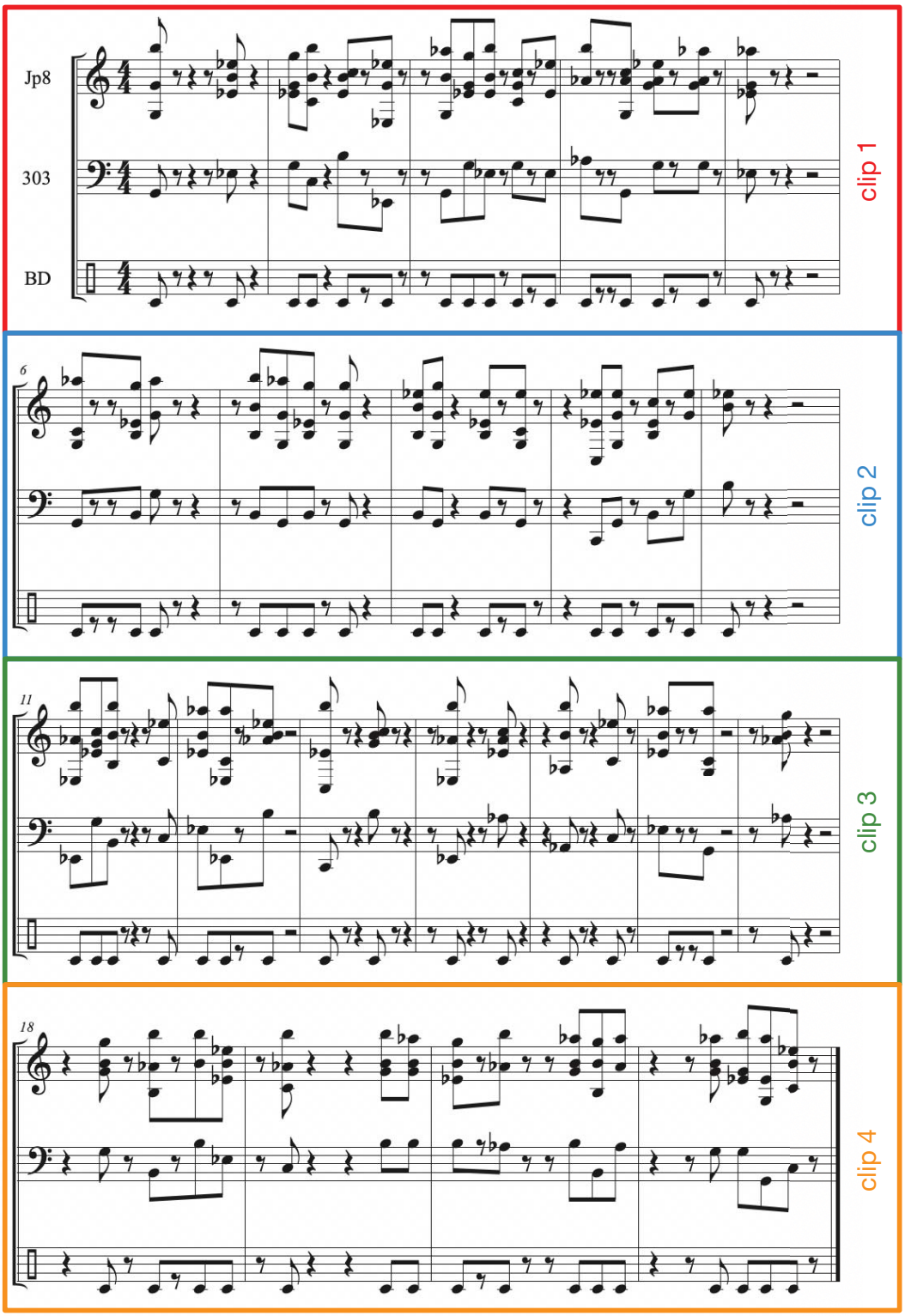


Figura 209: los doce clips juntos

Al concluir esta etapa, entonces, se clasifican los clips del track de bombo. Para esto se utiliza, en ambos ejes cartesianos, la medida cronotónica. Así el orden de los clips queda 1, 2, 4, 3. En este orden se copian y pegan en el mismo track pero la ventana de arreglo de *Live.* Lo mismo se hará con los tracks de bajo y de sintetizador, es decir, se copiarán y pegarán de acuerdo al orden obtenido en el bombo.

A continuación se genera, a través de *AMI*, cuatro clips para el track de hi-hat. En esta ocasión se utiliza, para el ritmo, el primer algoritmo de L-System y para las dinámicas se utiliza el generador llamado “Graphic” con grupos de 3, 2, 4 y 3 notas. El primer grupo tendrá una velocity de 118, el segundo una de 89, el tercero de 105 y el cuarto de 70. Este procedimiento, al crear grupos de dinámicas, resulta en un ritmo que se superpone al de las entradas de las notas. La altura, en este caso, será siempre la misma (FA#) ya que es un sonido no tónico. Así, entonces, se crean otros cuatro clips para el hi-hat y se clasifican de acuerdo a su valor cronotónico y su entropía en la dinámica. Entonces, como en este ejemplo se seleccionó el tercer clip como aquel con el que se compararán los otros, el orden queda 2, 3, 1, 4 y así se colocan en el track de este instrumento. Estos tracks se juntan en uno solo y se selecciona, en el *Clip View*, el modo loop para que este clip se repita tantas veces como sea necesario para que la duración del mismo sea idéntica a la partitura de la figura 209. Por último se agregaron otros instrumentos, pero sin utilizar *AMI*, entonces se creó una secuencia de redoblante (en 3/4 para agregar un elemento polirrítmico), un track de aro de redoblante (también en 3/4), un track de platillos tipo crash, uno de plato tipo ride y unos acordes que son ejecutados con un emulador de un Mellotron[107] con sonido de cuerdas. En la siguiente figura puede verse y escucharse el resultado final.

---

107 https://en.wikipedia.org/wiki/Mellotron

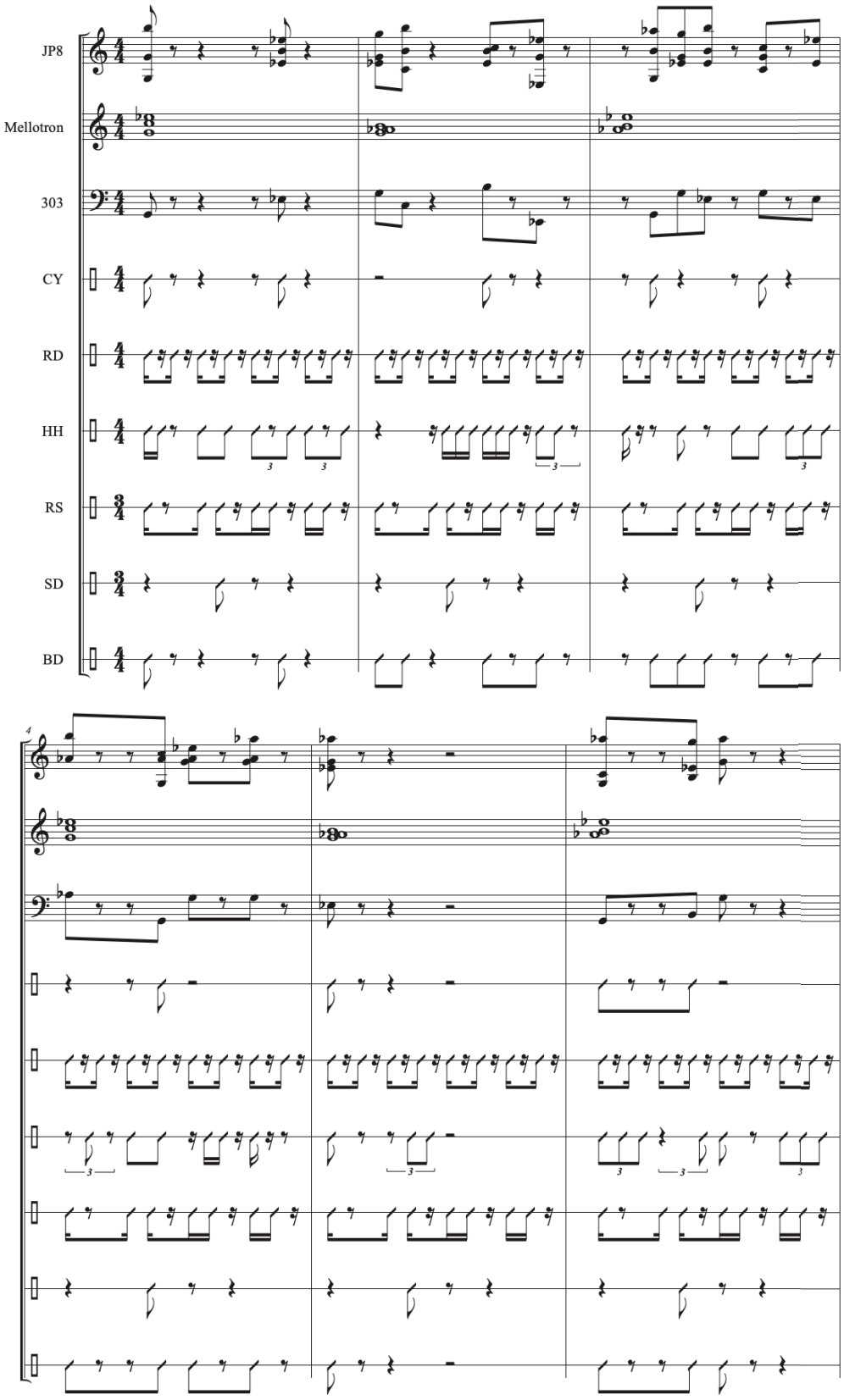


Figura 210: el primer caso de estudio

5.2 Segundo caso

Para este segundo caso se creará una partitura para un ensamble tipo Pierrot (flauta, clarinete, violín, violonchelo y piano). Se generaron ocho clips para cada instrumento.

Para las alturas se utilizaron, en los mismos clips de cada instrumento, el mismo Pitch Class Set:

Clip 1 = PCs 8-6

Clip 2 = PCs 8-10

Clip 3 = PCs 8-14

Clip 4 = PCs 8-16

Clip 5 = PCs 8-20

Clip 6 = PCs 8-22

Clip 7 = PCs 8-24

Clip 8 = PCs 8-28

Pero se utilizó un tipo diferente de algoritmo para cada instrumento:

Flauta: random walk

Clarinete: genético

Violín: caos logístico

Cello: aleatorio triangular

Piano: aleatorio lineal

Lo mismo se utilizó con los ritmos, es decir que cada instrumento fue generado con un único algoritmo, diferente para cada uno, y con una probabilidad de silencio del 30%. Para las dinámicas se utilizó, en todos los casos, caos logístico y aleatoriedad lineal. En la figura 211 se observa un ejemplo de los parámetros utilizados para generar el primer clip de la flauta y en la figura 212 los utilizados para el primer clip de clarinete:

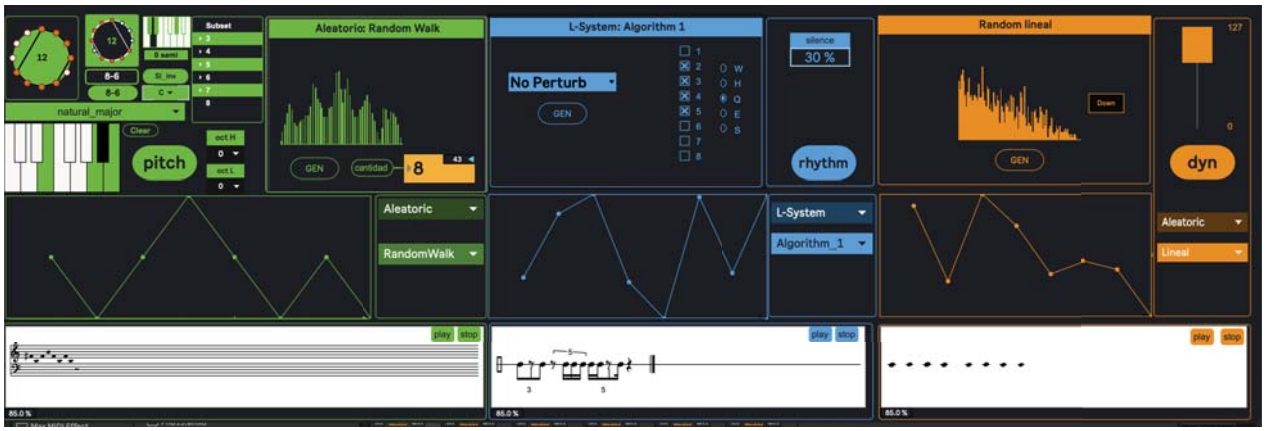


Figura 211: parámetros utilizados para generar el primer clip de flauta

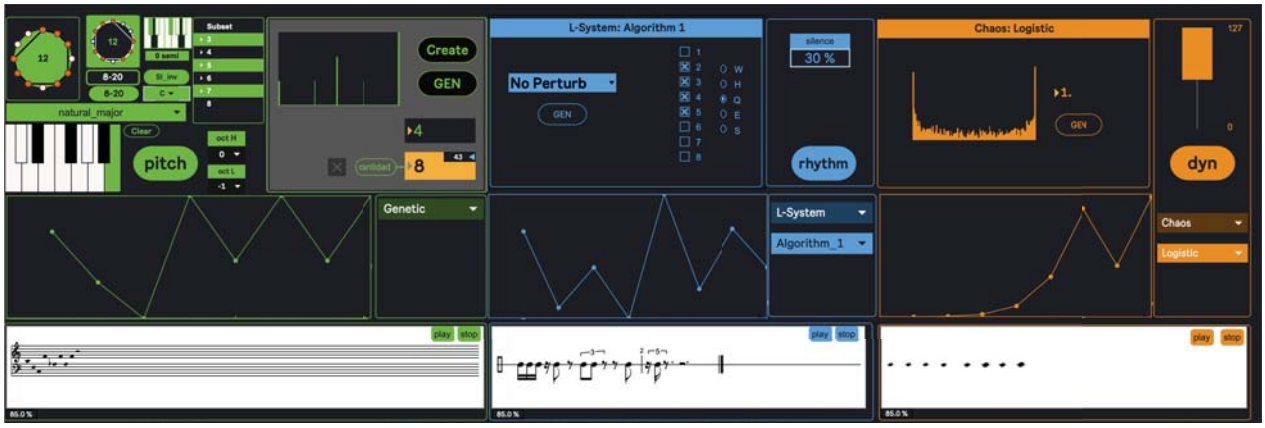


Figura 212: parámetros utilizados para generar el primer clip de clarinete

Una vez generados estos cuarenta clips, ocho por cada instrumento (figura 213), se procedió a clasificarlos. Para ello se utilizo como analizadores, la distancia de edición del perfil del pitch y la similitud cronotónica. Se eligieron estos dos tipos de análisis para obtener similitudes en el perfil de las secuencias y en los ritmos resultantes de la

generación algorítmica. Se le pidió al sistema que clasificara ocho clips tomando en cuenta todos los presentes en la sesión, es decir todos los instrumentos. (figura 214).

Figura 213: ocho clips en cada track

Figura 214: resultado de la primera clasificación

Una vez obtenida dicha clasificación se trasladaron los clips (en el orden obtenido) desde el modo de sesión al de arreglo, para de esta manera pegar dichos clips en el track de su instrumento correspondiente. Ver figura 215.

Figura 215: clips arreglados de acuerdo a su clasificación

Luego se analizaron los próximos ocho clips (los ya utilizados se borraron de la ventana de sesión figura 216) y se volvieron a copiar y pegar de acuerdo al resultado de la clasificación

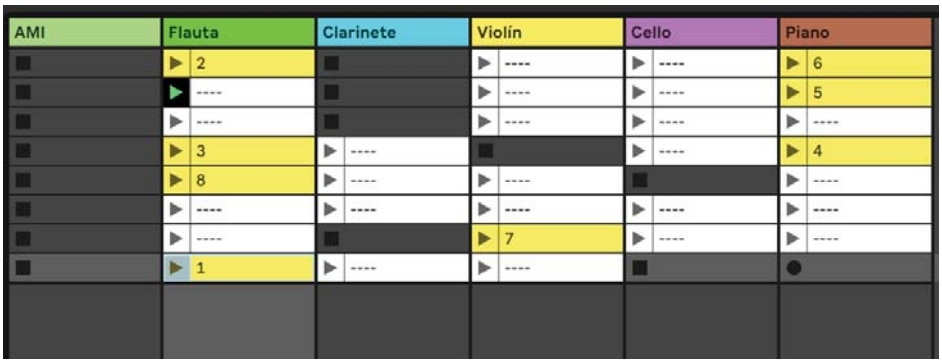


Figura 216: segunda clasificación

En la figura 217 se observa el resultado final al clasificar todos los clips.

Figura 217: secuencia final

Una vez obtenido este resultado se exportaron cada uno de los tracks como archivos MIDI para ser abiertos en un editor de partituras. Allí se modificaron las distancias entre cada secuencia de cada clip (respetando el orden obtenido en la clasificación) y se especificaron diferentes articulaciones en cada instrumento, trabajando así las características tímbrica de cada uno. En la figura 218 se observa la primera página[108] de esta partitura y se puede escuchar el resultado final donde se utilizan muestras de instrumentos acústicos.

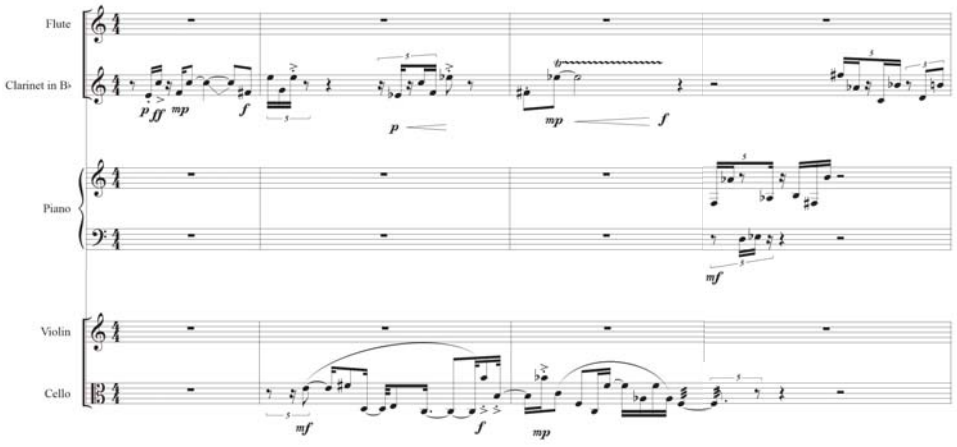


Figura 218: partitura obtenida

108 La partitura completa se anexa a esta tesis.

Conclusiones

Este trabajo fue concebido y llevado a término desde la creación musical con datos simbólicos y por lo tanto deja fuera un enorme campo de trabajo como lo es la creación a partir de técnicas de la música concreta. Podría argumentarse que *AMI* impone un tipo de composición “encasillada”, donde el ámbito es la nota, su altura, su duración, su dinámica. La plasticidad que otras estrategias de composición permiten, como en el caso de la composición de música electroacústica, no es una posibilidad que esta herramienta permita. También, y como consecuencia de lo antedicho, quedan fuera de este trabajo, entonces, dos parámetros muy importantes involucrados en la composición musical: el timbre y la especialización. Sin embargo, agregar estos dos parámetros puede ser parte de un trabajo posterior.

Respecto al mencionado “encasillamiento” se puede decir que lo que *AMI* genera son materiales con los que el compositor puede trabajar, quitando o agregando partes, reorganizado los datos obtenidos, modificando algunas secciones etc. Lo mismo ocurre con las diferentes clasificaciones que se pueden obtener del material. Como se menciona en la tesis, estas clasificaciones aportan datos objetivos pero no necesariamente “musicales” como puede ser el caso de clasificar diferentes secuencias según la desviación estándar de sus alturas. Sin embargo, y luego de hacer un uso extensivo de *AMI*, se concluye que resulta una herramienta muy útil para la creación de material “en crudo” que posea una lógica interna en cuanto al desarrollo de sus perfiles de alturas, ritmos y dinámicas y los algoritmos involucrados en dicha tarea.

La programación de la herramienta que se propuso crear aquí implicó la búsqueda de

soluciones a través de estrategias muy diversas: el uso de sistemas de notación musical tradicional, la creación de secuenciadores más o menos sofisticados, el almacenamiento de datos en bases externas e internas, el desarrollo de diferentes interfaces de usuario.

Los algoritmos escogidos resultaron muy propicios para la generación de alturas, sin embargo no lo son tanto para la generación rítmica. La necesidad de crear ritmos que sean legibles por un posible intérprete limita los posibles resultados que se pueden obtener. Sin embargo es factible pensar que se puede mejorar mucho esta parte para que los ritmos reflejen una mayor variedad según el algoritmo escogido para su generación. En cuanto a las dinámicas también existe un enorme espacio para mejorar su generación y crear perfiles de cambios más suaves y naturales. Aquí podría profundizarse la investigación sobre el uso de técnicas de la Inteligencia Artificial.

Algunas de las funciones que quedaron fuera de esta herramienta, principalmente por la necesidad de acotar el campo de trabajo, son la posibilidad de trabajar de manera polifónica, contar con un sistema más sofisticado para crear ritmos, utilizar escalas microtonales, utilizar estrategias de generación de datos que impliquen el uso de programación de tipo restrictiva (constraint programming) entre otras. Implementar estas otras funciones, será una tarea que se llevará a cabo en los meses (y quizá años) por venir.

Respecto a los tipo de analizadores implementados también aquí el campo de expansión es enorme. Este trabajo muestra que es posible desarrollar un sistema de generación y clasificación de datos musicales dentro de una plataforma comercial como lo es *Ableton Live*. El uso de *Max for Live* y el ejecutar la programación de una manera modular es un

gran acierto. Esto permite mejorar partes del código sin poner en peligro el resto del trabajo y, mas importante, permite agregar nuevas funciones y tipos de análisis de manera sencilla.

No es posible afirmar que el resultado final, es decir la herramienta *AMI*, sea la mejor estrategia ya que todo trabajo es perfectible. Sin embargo el resultado obtenido es lo suficientemente estable y amigable para el usuario como para, a partir de aquí, mejorar y acrecentar las posibilidades de esta herramienta.

Solo resta compartir *AMI* con la comunidad de creadoras y creadores musicales para que sean ellas y ellos los encargados de encontrar las flaquezas y las virtudes del sistema y puedan sugerir o directamente implementar mejorías a la herramienta.

Bibliografía

Agon, C., Assayag, G., & Bresson, J. (2006). *The OM composer's book 1*: Editions Delatour France/Ircam-Centre Pompidou.

Agostini, A., & Ghisi, D. (2012). Bach: An environment for computer-aided composition in max. Paper presented at the ICMC.

Aloupis, G., Fevens, T., Langerman, S., Matsui, T., Mesa, A., Nunez, Y., . . . Toussaint, G. (2006). Algorithms for computing geometric measures of melodic similarity. *Computer Music Journal, 30*(3), 67-76.

Alphonce, B. H. (1980). Music analysis by computer: A field for theory formation. *Computer Music Journal, 4*(2), 26 - 35. doi:papers3://publication/uuid/4C01462B-B39C-4479-8E61-5953CF3D9BC3

Alsop, R. (1999). Exploring the self through algorithmic composition. *Leonardo music journal*, 89-94.

Ames, C. (1987). Automated composition in retrospect: 1956-1986. *Leonardo*, 169-185.

Amram, M., Fisher, E., Gul, S., & Vishne, U. (2020). A Transformational Modified Markov Process for Chord-Based Algorithmic Composition. *Mathematical and Computational Applications, 25*(3), 43.

Anders, T., & Miranda, E. R. (2011). Constraint programming systems for modeling music theories and composition. *ACM Computing Surveys (CSUR), 43*(4), 1-38.

Andrés, M. L. M. (1996). Algoritmos de búsqueda de vecinos más próximos en espacios métricos. (PhD),

Apel, W. (2003). *The Harvard dictionary of music* (Vol. 16): Harvard University Press.

Ariza, C. (2011). Two pioneering projects from the early history of computer-aided algorithmic composition. *Computer Music Journal, 35*(3), 40-56.

Assayag, G. (1998). *Computer assisted composition today.* Paper presented at the 1st symposium on music and computers, Corfu.

Assayag, G., Agon, C., Fineberg, J., & Hanappe, P. (1997). *An object oriented visual environment for musical composition.* Paper presented at the ICMC: International Computer Music Conference.

Assayag, G., Rueda, C., Laurson, M., Agon, C., & Delerue, O. (1999). Computer-assisted composition at IRCAM: From PatchWork to OpenMusic. *Computer Music Journal, 23*(3), 59-72.

Baboni-Schilingi, J., & Voisin, F. (1999). *Morphologie: Documentation OpenMusic*.

Baker, J. E. (1987, 1987). *Reducing bias and inefficiency in the selection algorithm.* Paper presented at the Genetic Algorithms and their applications: Proceedings of the second

International Conference on Genetic Algorithms.

Bel, B., & Vecchione, B. (1993). Computational Musicology. *Computers and the Humanities, 27*(1).

Borges, J. L. (1960). El idioma analítico de John Wilkins. In *Otras Inquisiciones*. Buenos Aires: Emecé Editores.

Bresson, J., Bouche, D., Carpentier, T., Schwarz, D., & Garcia, J. (2017). Next-generation Computer-aided Composition Environment: A new implementation of OpenMusic. Paper presented at the International Computer Music Conference (ICMC'17).

Brown, A. R. (1999). An introduction to music analysis with computer. *XArt Online Journal, 4*(1).

Budón, O. (2000). Composing with objects, networks, and time scales: an interview with Horacio Vaggione. *Computer Music Journal, 24*(3), 9-22.

Byrd, D. (1977). An integrated computer music software system. *Computer Music Journal*, 55-60.

Carvalho, N., & Bernardes, G. SyVMO: Synchronous Variable Markov Oracle for modeling and predicting multi-part musical structures.

Cetta, P. C. (2018). Fundamentos de composición musical asistida en el entorno de

programación OpenMusic.

Cetta, P. C., & Di Liscia, O. (2010). *Elementos de contrapunto atonal*: Educa.

Chen, W., Keast, J., Moody, J., Moriarty, C., Villalobos, F., Winter, V., . . . Wang, J. (2019). *Data Usage in MIR: History & Future Recommendations.* Paper presented at the ISMIR.

Chomsky, N. (1957). *Syntactic Structures*. Berlin: De Gruyter Mouton.

Cointe, P., & Rodet, X. (1984). *Formes: an object and time oriented system for music composition and synthesis.* Paper presented at the Proceedings of the 1984 ACM Symposium on LISP and Functional Programming.

Colasanto, F. (2010). *Max/MSP: guía de programación para artistas. Volumen 1*: Centro Mexicano para la Música y las Artes Sonoras (CMMAS).

Collins, N. (2018). Origins of Algorithmic Thinking in Music. In A. M. a. R. Dean (Ed.), *The Oxford Handbook of Algorithmic Composition* (pp. 67–78). New York, NY: Oxford University Press.

Cope, D. (1987). Experiments in Music Intelligence (EMI). *ICMC*. doi:papers3://publication/uuid/3947F9BA-D38C-4199-A2EB-EB813221F15F

Cope, D. (1992). Computer modeling of musical intelligence in EMI. *Computer Music Journal, 16*(2), 69-83.

Courtot, F. (1992). CARLA: Acquisition et induction sur le matériau compositionnel. Rennes 1,

Darwin, C. (1859). On the Origin of Species by Means of Natural Selection, Or, The Preservation of Favoured Races in the Struggle for Life.

Dewdney, A. K. (1987). Computer Recreations. *Scientific American, 257*(1), 108-111.

Di Liscia, O. P. (2011). Medidas de similitud entre sucesiones ordenadas de grados cromáticos.

Dodge, C., & Jerse, T. A. (1985). Computer music: synthesis, composition, and performance. New York, NY: Schirmer.

Doraisamy, S., & Rüger, S. M. (2002). A Comparative and Fault-tolerance Study of the Use of N-grams with Polyphonic Music. Paper presented at the ISMIR.

Downie, J. S. (1999). Evaluating a simple approach to music information retrieval: Conceiving melodic n-grams as text: Citeseer.

Downie, J. S. (2004). The scientific evaluation of music information retrieval systems: Foundations and future. *Computer Music Journal, 28*(2), 12-23.

Duarte, A. E. L. (2020). Algorithmic interactive music generation in videogames. *SoundEffects-An Interdisciplinary Journal of Sound and Sound Experience, 9*(1), 38-59.

Eckel, G. (1988). About PreFORM. 1-16. doi:papers3://publication/uuid/60B22984-072D-4E2D-BF58-CD67D966FF0D

Ens, J., & Pasquier, P. (2020). Quantifying musical style: Ranking symbolic music based on similarity to a style. *arXiv preprint arXiv:2003.06226*.

Erickson, R. F. (1968). Musical analysis and the computer: a report on some current approaches and the outlook for the future. *Computers and the Humanities*, 87-104.

Essl, K. (2007). Algorithmic composition. In N. C. a. J. d'Escriván (Ed.), *The Cambridge Companion to Electronic Music*. New York: Cambridge University Press.

Estrada, J., & Gil, J. (1984). Música y teoría de grupos finitos (3 variables booleanas).

Feldman, D. P. (2012). Chaos and Fractals: An Elementary Introduction. UK: Oxford University Press.

Fernández, J. D., & Vico, F. (2013). Ai methods in algorithmic composition: A comprehensive survey. *Journal of Artificial Intelligence Research 48*.

Forte, A. (1973). The Structure of Atonal Music: Yale University Press.

Fridenfalk, M. (2015). *Algorithmic music composition for computer games based on L-system.* Paper presented at the 2015 IEEE Games Entertainment Media Conference

(GEM).

Frieler, K. (2006). Generalized N-gram measures for melodic similarity. In *Data science and classification* (pp. 289-298): Springer.

Gardner, M. (1971). On cellular automata, self-reprodiction, the garden of eden and the game "life". *Scientific American*, 112-117.

Gerzso, A. (1984). Reflections on Répons. *Contemporary Music Review, 1*(1), 23-34.

Gillick, J., Tang, K., & Keller, R. M. (2010). Machine Learning of Jazz Grammars. *Computer Music Journal, 34*(3). doi:papers3://publication/uuid/65D639E6-1D14-476A-B213-3473B9253C9D

Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Alabama, USA: Addison-Wesley Professional.

Gómez Marín, D., Jordà Puig, S., & Boyer, H. (2015). *Pad and Sad: Two awareness-Weighted rhythmic similarity distances.* Paper presented at the Müller M, Wiering F, editors. Proceedings of the 16th International Society for Music Information Retrieval (ISMIR) Conference; 2015 Oct 26-30; Málaga, Spain. Canada: International Society for Music Information Retrieval; 2015.

Hed, S., Gjerdingen, R. O., & Levin, D. (2015). Pitch-and-rhythm interrelationships and musical patterns: Analysis and modelling by subdivision schemes. *Journal of Mathematics*

and Music, 9(1), 45-73.

Hedelin, F. (2008). Formalising form: An alternative approach to algorithmic composition. *Organised sound, 13*(3), 249-257.

Hedges, S. A. (1978). Dice music in the eighteenth century. *Music & Letters, 59*(2), 180-187.

Heo, H., Kim, H. J., Kim, W. S., & Lee, K. (2017). Cover Song Identification with Metric Learning Using Distance as a Feature. Paper presented at the ISMIR.

Hewlett, W. B., & Selfridge-Field, E. (1991). Computing in musicology, 1966–91. *Computers and the Humanities, 25*(6), 381-392.

Hofmann-Engl, L. (2002). *Rhythmic similarity: A theoretical and empirical approach.* Paper presented at the Proceedings of the Seventh International Conference on Music Perception and Cognition.

Holtzman, S. R. (1981). Using generative grammars for music composition. *Computer Music Journal, 5*(1). doi:papers3://publication/doi/10.2307/3679694

Hoos, H. H., Renz, K., & Görg, M. (2001). GUIDO/MIR-an Experimental Musical Information Retrieval System based on GUIDO Music Notation. Paper presented at the ISMIR.

Ikeda, K. (1979). Multiple-valued stationary state and its instability of the transmitted light by a ring cavity system. *Optics Communications, 30*(2), 257-261. doi:10.1016/0030-4018(79)90090-7

Isaacson, E. J. (1990). Similarity of interval-class content between pitch-class sets: the IcVSIM relation. *Journal of Music Theory*, 1-28.

Jacob, B. (1996). Algorithmic composition as a model of creativity. *Organised Sound. Cambridge University Press, 1*(3).

Kassler, M. (1966). Toward musical information retrieval. *Perspectives of New Music*, 59-67.

Knotts, S., & Collins, N. (2018). Introduction to Algorithmic EDM. *Dancecult: Journal of Electronic Dance Music Culture, 10*(1).

L'Ecuyer, P. (2017). *History of uniform random number generation.* Paper presented at the 2017 Winter Simulation Conference, Las Vegas.

Langston, P. (1988). Six techniques for algorithmic music composition. *15th International Computer Music Conference (ICMC)*.

Lerdahl, F., & Jackendoff, R. (1983). *A Generative Theory of Tonal Music*. Cambridge, MA.: MIT press.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. Paper presented at the Soviet physics doklady.

Li, H., Tang, Z., Fei, X., Chao, K.-M., Yang, M., & He, C. (2017). A Survey of Audio MIR Systems, Symbolic MIR Systems and a Music Definition Language Demo-System.

Li, W., & Nordahl, M. G. (1992). Transient behavior of cellular automaton rule 110. *Physics Letters A, 166*(5-6), 335-339. doi:10.1016/0375-9601(92)90718-2

Lin, Y.-T., Liu, I.-T., Jang, J.-S. R., & Wu, J.-L. (2015). Audio musical dice game: A user-preference-aware medley generating system. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 11*(4), 1-24.

Lindenmayer, A. (1968). Mathematical models for cellular interactions in development. *Journal of Theoretical Biology*(18). doi:papers3://publication/uuid/E6EB0DBF-BAAB-496B-A160-CAF706F44027

Lippe, C. (1991). Real-time computer music at IRCAM. *Contemporary Music Review, 6*(1), 219-224.

Lopez-Rincon, O., Starostenko, O., & Ayala-San Martín, G. (2018). *Algoritmic music composition based on artificial intelligence: A survey.* Paper presented at the 2018 International Conference on Electronics, Communications and Computers (CONIELECOMP).

Loy, G. (2011). Musimathics: the mathematical foundations of music (Vol. 1): MIT press.

Manning, P. (2004). *Electronic and Computer Music*. New York: Oxford University Press.

Manousakis, S. (2006). Musical L-systems.

Martinez, G., McIntosh, H., Mora, J., Martinez, G., McIntosh, H., & Mora, J. (2006). Gliders in Rule 110. *International Journal of Unconventional Computing, 2*(1), 1-49.

Mazurowski, Ł. (2012). *Computer models for algorithmic music composition.* Paper presented at the 2012 federated conference on computer science and information systems (fedcsis).

McCartin, B. J. (1998). Prelude to musical geometry. *The College Mathematics Journal, 29*(5), 354-370.

McCartney, J. (1996). *SuperCollider: a new real time synthesis language.* Paper presented at the Proc. International Computer Music Conference (ICMC'96).

McCormack, J. (1996). *Grammar based music composition.* Paper presented at the Complex Systems Conference 96: From Local Interactions to Global Phenomena, Amsterdam.

McHard, J. L. (2008). The Future of Modern Music: A Philosophical Exploration of Modernist Music in the 20th Century and Beyond.

McIlwain, P. A., & McCormack, J. P. (2005). *Design issues in musical composition networks.* Paper presented at the Australasian Computer Music Conference 2005.

McIntosh, H. V. (2010). Game of Life Cellular Automata. London: Springer London.

McLean, A., & Dean, R. T. (2018a). Musical algorithms as tools, languages, and partners. *The Oxford Handbook of Algorithmic Music*, 1.

McLean, A., & Dean, R. T. (2018b). The Oxford handbook of algorithmic music: Oxford University Press.

Michelitsch, M., & Rössler, O. E. (1992). Chaos and graphics the " burning ship " and its quasi-julia sets. *Computers & Graphics, 16*(4), 287-290.

Miranda, E. R. (2001). Composing Music with Computers: Focal Press.

Miranda, E. R., & Biles, J. A. (2007). *Evolutionary Computer Music*. London: Springer-Verlag.

Mitchell, M. (1996). *Introduction to genetic algorithms*. Cambridge, Massachusetts: The MIT Press.

Mor, B., Garhwal, S., & Kumar, A. (2019). A systematic literature review on computational musicology. *Archives of Computational Methods in Engineering*, 1-15.

Morgan, N., & Podrazik, J. (2020). Quick Guide. In *Introduction to OPUSMODUS* (pp. 1-79).

Moritz, M., Heard, M., Kim, H.-W., & Lee, Y. S. (2020). Invariance of edit-distance to tempo in rhythm similarity. *Psychology of Music*, 0305735620971030.

Müllensiefen, D., & Frieler, K. (2006). Evaluating different approaches to measuring the similarity of melodies. In *Data Science and Classification* (pp. 299-306): Springer.

Müllensiefen, D., & Pendzich, M. (2009). Court decisions on music plagiarism and the predictive value of similarity algorithms. *Musicae Scientiae, 13*(1_suppl), 257-295.

Neary, T., & Woods, D. (2006). *P-completeness of cellular automaton rule 110.* Paper presented at the International Colloquium on Automata, Languages, and Programming., Berlin.

Neubarth, K., Bergeron, M., & Conklin, D. (2011). Associations between Musicology and Music Information Retrieval. Paper presented at the ISMIR.

Nierhaus, G. (2009). *Algorithmic Composition*. Vienna: Springer Science & Business Media.

OMaidin, D. (1998). A geometrical algorithm for melodic difference. *Computing in Musicology, 11*, 65-72.

Park, S., Kwon, T., Lee, J., Kim, J., & Nam, J. (2019). A Cross-Scape Plot Representation for Visualizing Symbolic Melodic Similarity. Paper presented at the ISMIR.

Poli, R., Langdon, W. B., & McPhee, N. F. (2008). *A Field Guide to Genetic Programing*: Creative Commons.

Pople, A. (1992). Computer music and computer-based musicology. *Computers & Education, 19*(1-2), 173-182.

Post, O., & Toussaint, G. (2011). The edit distance as a measure of perceived rhythmic similarity.

Prusinkiewicz, P. (1986). *Score generation with L-systems.* Paper presented at the International Computer Music Conference, Den Haag, The Netherlands.

Prusinkiewicz, P., & Lindenmayer, A. (2012). *The Algorithmic Beauty of Plants*: Springer Science & Business Media.

Puckette, M. (1986). Interprocess communication and timing in real-time computer music performance: Ann Arbor, MI: Michigan Publishing, University of Michigan Library.

Puckette, M. (1991). Combining event and signal processing in the MAX graphical programming environment. *Computer Music Journal, 15*(3), 68-77.

Roads, C. (2015). Composing electronic music: a new aesthetic: Oxford University Press, USA.

Roads, C., & Strawn, J. (1996). *The computer music tutorial*: MIT press.

Roads, C., & Wieneke, P. (1979). Grammars as Representations for Music. *Computer Music Journal, 3*(1), 48-48. doi:10.2307/3679756

Ross, A. (2007). The rest is noise: Listening to the twentieth century: Macmillan.

Rössler, O. E. (1976). An equation for continuous chaos. *Physics Letters, 57*(5), 397-398. doi:10.1016/0375-9601(76)90101-8

Salazar, C., & Castillo, S. d. (2018). Fundamentos básicos de estadística. In: Editor no identificado.

Savery, R. (2018). An interactive algorithmic music system for edm. *Dancecult: Journal of Electronic Dance Music Culture, 10*(1).

Schuitemaker, N. (2020). An Analysis of Melodic Plagiarism Recognition using Musical Similarity Algorithms.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell system technical journal, 27*(3), 379-423.

Shukla, A., Pandey, H. M., & Mehrotra, D. (2015). *Comparative review of selection*

techniques in genetic algorithm. Paper presented at the 1st International Conference on Futuristic Trends in Computational Analysis and Knowledge Management, Nodia, India.

Simoni, M. (2003). Algorithmic composition: a gentle introduction to music composition using common LISP and common music: Michigan Publishing, University of Michigan Library.

Siphocly, N. N. E.-H., El-Sayed Salem, Abdel-Badeeh. (2021). Top 10 Artificial Intelligence Algorithms in Computer Music Composition. *journal.uob.edu.bh*. doi:papers3://publication/doi/10.12785/ijcds/100138

Spiegel, L. (2018). Thoughts on Composing with Algorithms. *The Oxford Handbook of Algorithmic Music*, 105.

Steele, G. L., & Gabriel, R. P. (1996). The evolution of Lisp. *History of programming languages---II*, 233-330.

Stein, P. R. (1987). Iteration of maps, strange attractors, and number theory - An Ulamian potpourri. *Los Alamos Science Special*.

Supper, M. (2001). A few remarks on algorithmic composition. *Computer Music Journal, 25*(1), 48-53.

Typke, R., Wiering, F., & Veltkamp, R. C. (2007). Transportation distances and human perception of melodic similarity. *Musicae Scientiae, 11*(1_suppl), 153-181.

Umbarkar, A. J., & Sheth, P. D. (2015). Crossover Operators in Genetic Algorithms: a Review. *ICTACT Journal on Soft Computing, 6*(1), 1083-1092. doi:10.21917/ijsc.2015.0150

Urbano, J., Lloréns, J., Morato, J., & Sánchez-Cuadrado, S. (2010). *Melodic similarity through shape similarity.* Paper presented at the International Symposium on Computer Music Modeling and Retrieval.

Vaggione, H. (1970). Obtención de formas musicales a través de la codificación digital de textos:(Seminario de Música). *Boletín del Centro de Cálculo de la Universidad de Madrid*(12), 18-34.

Vázquez, J. I., & Oliver, J. (2008). Evolución de autómatas celulares utilizando algoritmos genéticos. *Universidad de Deusto.* doi:papers3://publication/uuid/849A9D28-92AA-4145-8E52-F3F12B177904

Velardo, V., Vallati, M., & Jan, S. (2016). Symbolic melodic similarity: State of the art and future challenges. *Computer Music Journal, 40*(2), 70-83.

Volk, A., Wiering, F., & Kranenburg, P. (2011). *Unfolding the potential of computational musicology.* Paper presented at the Proceedings of the 13th International Conference on Informatics and Semiotics in Organisations.

Wolfram, S. (1985). Two-Dimensional Cellular Automata. *Cellular Automata and Complexity, 38*(March), 211-249. doi:10.1201/9780429494093-6

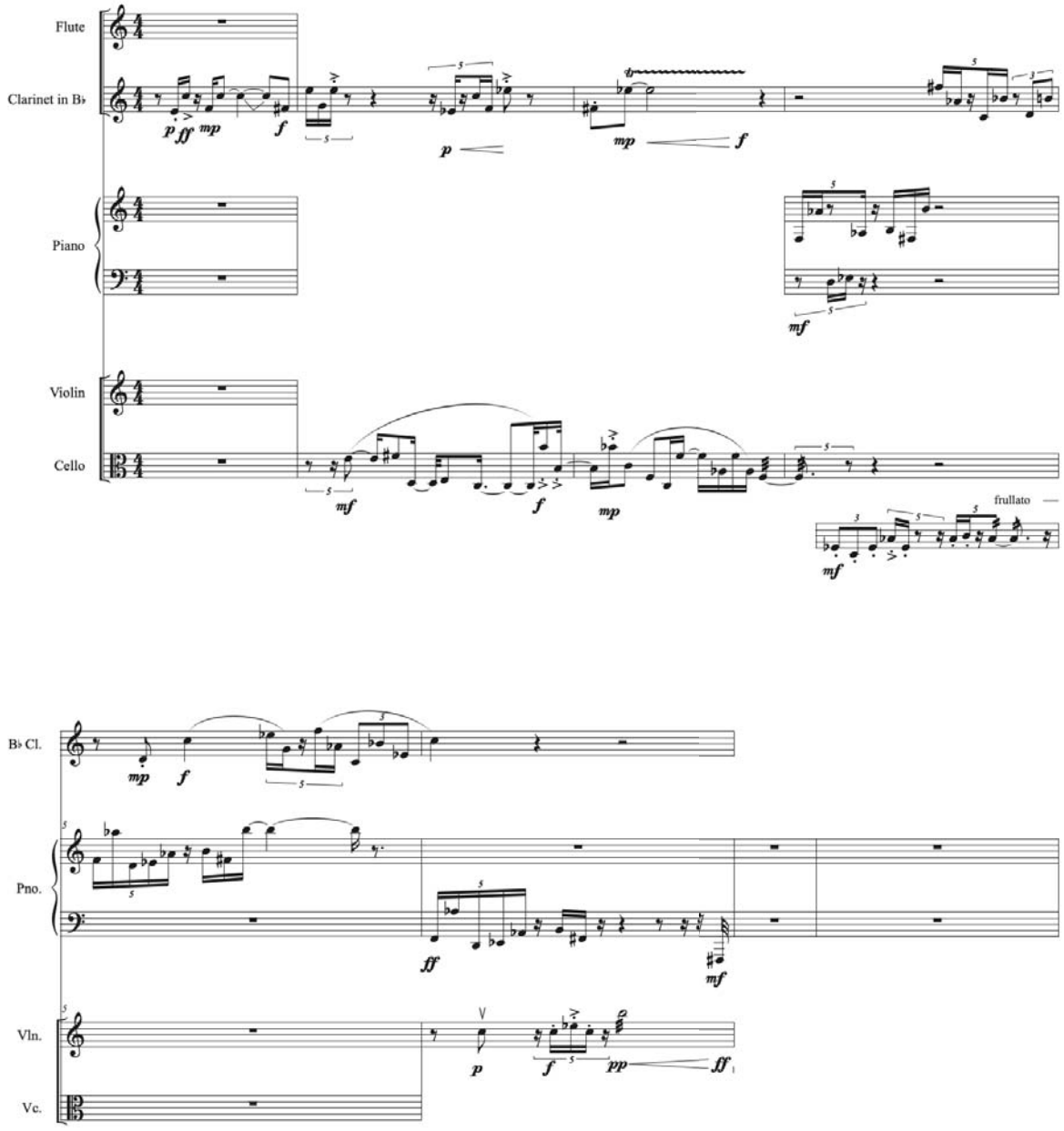
Wolkowicz, J., & Kešelj, V. (2011). Text information retrieval approach to music information retrieval. *Music Information Retrieval Evaluation eXchange*.

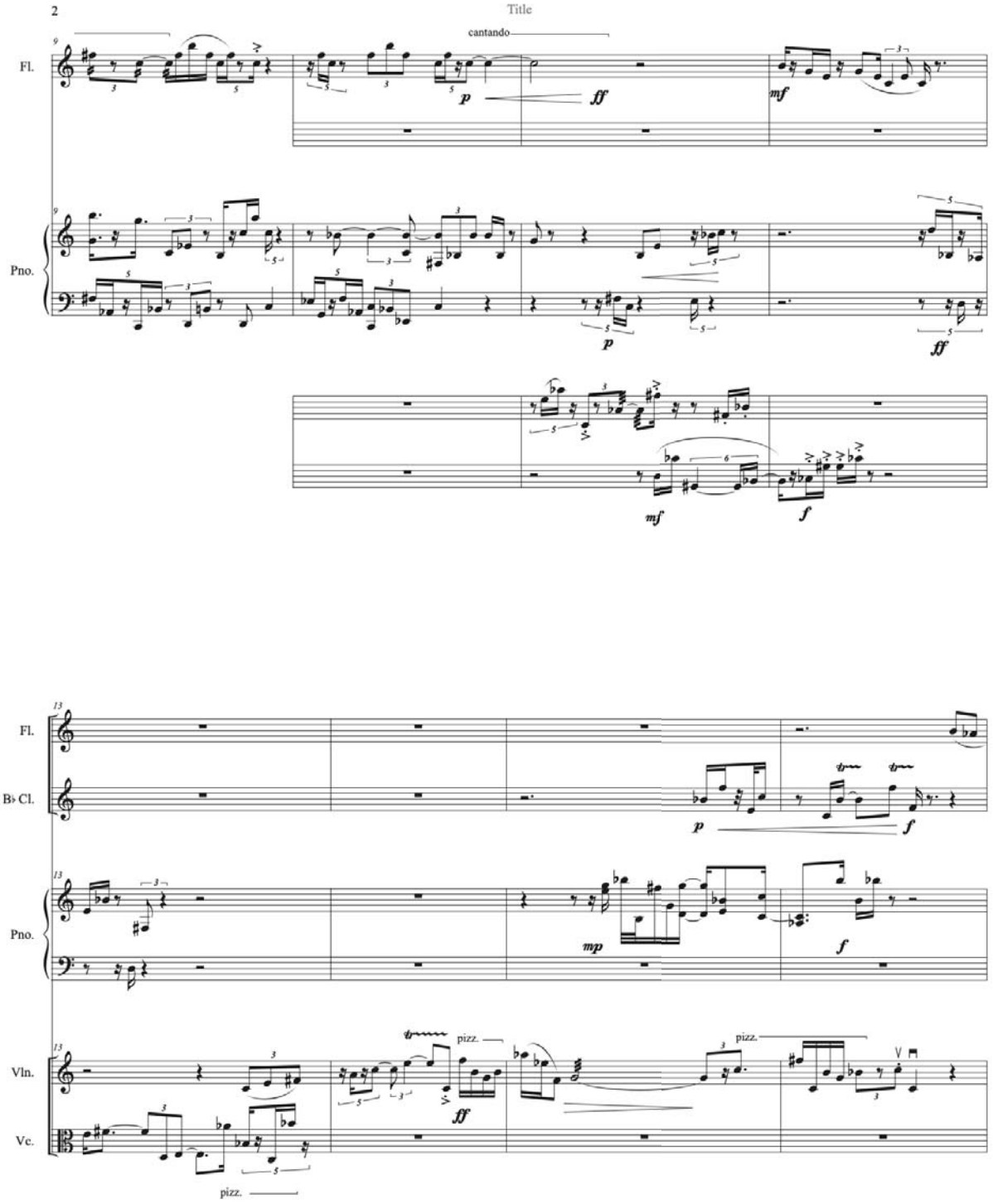
Xenakis, I. (1963). Musiques formelles : nouveaux principes formels de composition musicale. *La Revue Musicale*.

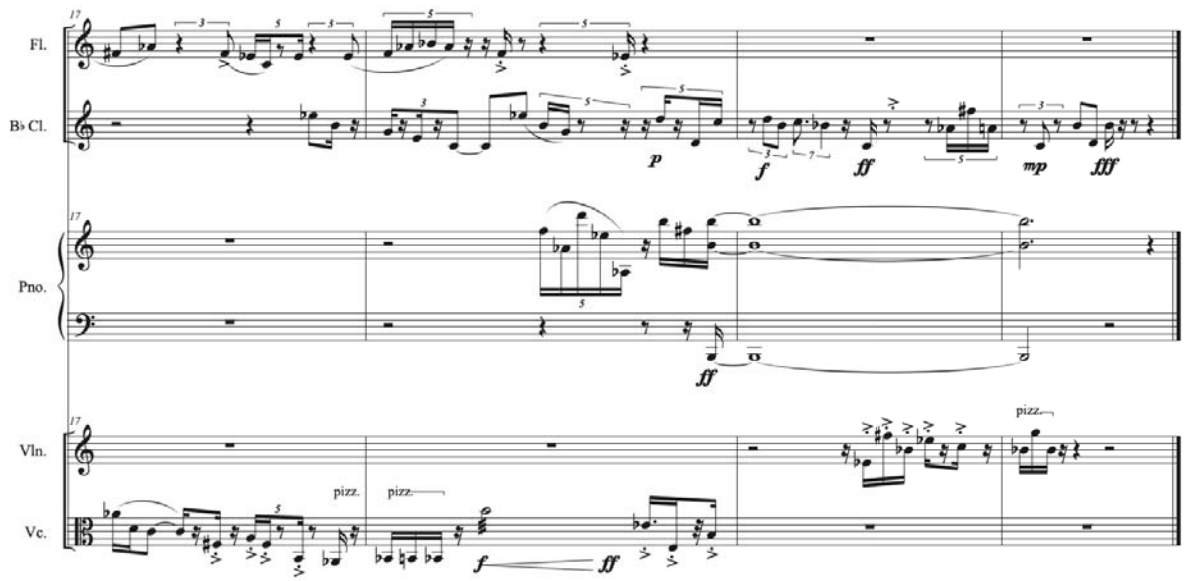
Xenakis, I. (1971). Formalized Music: Thought and Mathematics in Composition (Vol. 30): Indiana University Press.

Anexo

Partitura del caso 2




2
Title
cantando
Fl.
Pno.
Fl.
B♭ Cl.
Pno.
Vln.
Vc.
pizz.
pizz.
pizz.


17
Fl.
B♭ Cl.
p
f
ff
mp
fff
17
Pno.
ff
17
Vln.
Vc.
pizz.
pizz.
pizz.
f
ff