Universidad Nacional Autónoma de México

Facultad de Ingeniería

Análisis de replicación de sonidos cortos a partir de síntesis de primitivas utilizando algoritmos genéticos

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

Ingeniero en Computación

PRESENTA:

Ricardo Falcón Pérez

DIRECTOR DE TESIS:
Dr.José Abel Herera Camacho

México, D.F., 2016

Declaración de autenticidad

Por la presente declaro que, salvo cuando se haga referencia específica al trabajo de otras personas, el contenido de esta tesis es original y no ha sido presentado total o parcialmente para su consideración para cualquier otro título o grado en esta o cualquier otra Universidad. Esta tesis es resultado único de mi propio trabajo y no incluye resultados de algún trabajo realizado en colaboración, excepto cuando se indica específicamente en el texto.

Ricardo Falcón Pérez. México, D.F., 2016

Índice general

Índice de figuras

Índice de tablas

Índice de códigos

Capítulo 1

Introducción

1.1. Definición del problema

Los algoritmos genéticos son algoritmos para optimización que tienen la capacidad de evolucionar, o modificar las condiciones de operación a medida que se van ejecutando. Estos algoritmos ofrecen una ventaja importante para resolver problemas complejos donde intervienen varias variables independientes, ya que automáticamente gravitan hacia soluciones cada vez más eficientes sin el costoso proceso de analizar soluciones que tienen poca probabilidad de ser exitosas, al mismo tiempo que evitan soluciones no óptimas con máximos ó mínimos locales. Por esto, los algoritmos genéticos y otros métodos de cómputo evolutivo han sido estudiados rigurosamente y aplicados a diversos campos, como el plegamiento de proteínas, diseño de antenas y procesamiento de imágenes.

El concepto de algoritmos genéticos es sencillo, sin embargo la implementación es compleja porque existe una gran cantidad de parámetros que son necesarios definir, y cada problema específico responde de manera diferente a la configuración de estos parámetros. Una implementación de algoritmos genéticos aplicados a procesamiento de imágenes, es la replicación (copia) de una imagen base a partir de una combinación de polígonos de un solo color, dibujados en cierto orden. Con esta aplicación es posible obtener imágenes, que a juicio de un ser humano, tienen alto grado de parentesco a la imagen base, a pesar de que matemáticamente, las imágenes son muy diferentes.

Para señales de audio, existen pocas aplicaciones de cómputo evolutivo con el propósito de replicar sonidos existentes, y por tanto no hay mucho conocimiento empírico de como configurar los parámetros evolutivos para optimizar implementaciones de algoritmos genéticos a audio. Por esto es necesario diseñar e implementar una aplicación para poder replicar señales de audio a partir de una combinación de señales primitivas, y poder evaluar cómo influye la configuración de los parámetros evolutivos. Esto generará un precedente para el análisis de audio con cómputo evolutivo, y facilitará futuros estudios de audio con estos algoritmos.

Existe otro beneficio de esta aplicación en forma de compresión de audio. Ya que la señal replicada será generada a partir de la combinación de varias señales sencillas, el almacenamiento de las propiedades de estas señales será menor que almacenar la información completa de la señal original.

1.2. Objetivos

El principal propósito es diseñar e implementar un programa de cómputo que sea capaz de duplicar una señal de audio corta utilizando algoritmos genéticos y cómputo evolutivo. Para después poder analizar como influyen los parámetros del algoritmo genético y como afectan los resultados y al audio en general. Además se busca proveer un breve marco teórico que explique las bases de los algoritmos genéticos y el estado del arte de estos cuando se aplican a señales de audio y música, incluyendo también un repaso de procesamiento de audio y puntos importantes para comprender la percepción de timbre.

Adicionalmente, para completar estas tareas, primero se estudian funciones de distancia de similitud de audio y se propone una síntesis básica basada en primitivas. Estos conceptos alimentan el sistema de algoritmos genéticos.

Finalmente, los resultados serán analizados desde los puntos de vista objetivo y subjetivo. Es decir se interpretan los datos obtenidos por sus valores absolutos y se cualifica la calidad de los sonidos generados.

1.3. Metodología

La metodología que se empleó es teórica y experimental. Para la parte teórica se revisó el estado del arte de los algoritmos genéticos para identificar los últimos desarrollos del área y qué técnicas son las más apropiadas para manipular. Para la parte experimental, se tomó como ejemplo una implementación existente de algoritmos genéticos aplicados a replicación de imágenes y se realizó una implementación del mismo concepto pero aplicado a replicación de audio, específicamente de señales de audio cortas de carácter general.

Para la replicación de audio se utilizó una forma sencilla para generar los sonidos, ya que el objetivo es analizar cómo se desempeñan los algoritmos genéticos en la replicación de audio en general y no en alguna técnica específica de síntesis. Para esto se usaron combinaciones de señales primitivas de audio (cómo senoidales, triagulares, cuadradras y diente de sierra) y una envolvente de amplitud. El proceso siguió los siguientes pasos:

- Se ejecutaron experimentos usando algoritmos genéticos aplicados a imágenes para conocer el comportamiento de esta técnica.
- Se construyó un programa de computadora en C# para poder realizar experimentos de algoritmos genéticos sobre señales de audio. Para esta aplicación se definieron que parámetros evolutivos son los más importantes para ser configurables en las pruebas, y el método particular de síntesis que se usó para generar las señales de audio.
- Se ejecutaron diversos experimentos utilizando la aplicación y se analizaron los resultados.

Para el análisis de los resultados, se consideran cifras descriptivas, como lo son los indicadores de los procesos de evolución, los espectrogramas de las señales obtenidas, y las mediciones de distancia. Además, también se incluyen comparaciones subjetivas de

las señales generadas, basadas en la percepción de las mismas. Durante este trabajo se realizaron dos tipos de experimentos:

- Propuestas y análisis de diferentes funciones de distancia para medir la similitud de señales de audio enfocada a la percepción del timbre.
- Replicación de señales de audio cortas utilizando la aplicación construida documentando tanto el desempeño del algoritmo genético como la calidad de los resultados generados.

1.4. Contribución y Relevancia

Como principal contribución este trabajo aporta una aplicación eficiente, expansible y de fácil uso para hacer análisis de audio utilizando algoritmos genéticos. En su versión inicial la aplicación es suficientemente robusta para hacer análisis básicos, con diversos parámetros configurables y herramientas para compartir resultados y ver gráficas que explican el proceso de evolución. Esta aplicación es además expansible por lo que trabajos futuros podrán manipularla para introducir conceptos más complejos, ya sean algoritmos evolutivos más elaborados o una síntesis de audio menos general y más desarrollada.

El estudio sobre funciones de distancia identifica las propiedades de estas funciones y cómo se comportan cuando son aplicadas a señales de sonidos cortos. Abriendo la puerta para que estudios más intensivos comparen estos resultados con otros tipos de señales de audio, como la voz y música.

Por último, al comparar los sonidos generados con los originales se identifican las fortalezas y debilidades de la síntesis propuesta, y la relación que tienen los parámetros de esta con el proceso de evolución.

1.5. Estructura de la tesis

Este trabajo está dividido en 6 capítulos que cubren las 4 etapas de desarrollo. El capítulo 2 corresponde a al marco teórico donde se cubren de los conceptos básicos del cómputo evolutivo y los algoritmos genéticos, además de un breve panorama sobre el estado del arte de la aplicación de algoritmos genéticos a señales de audio y música.

El capítulo 3 incluye el análisis de un caso de estudio de una aplicación que utiliza algoritmos genéticos aplicados a replicación de imágenes. Este caso de prueba sirvió como ejemplo e inspiración para el desarrollo de la tesis. Se explica cómo está construido, las características principales y se ejecutaron algunos experimentos básicos para conocer su funcionamiento.

En el capítulo 4 comprende el diseño y construcción del proyecto así como de los experimentos realizados. Aquí se muestra tanto los componentes técnicos de la plataforma elaborada cómo la justificación teórica de algunas decisiones tomadas. Para los experimentos se describen las condiciones generales, el comportamiento esperado y las señales particulares utilizadas para cada prueba. Al igual que en el capítulo anterior, para las operaciones más relevantes se incluyen secciones de código fuente.

El capítulo 5 contiene los resultados de los experimentos descritos en el capítulo 4. No se muestran todos los datos obtenidos sino que se condensan en tablas, figuras y

espectrogramas ya sea de casos representativos o el promedio de las ejecuciones. Además se hace un análisis de estos resultados, destacando lo más sobresaliente.

Por último, en el capítulo 6 se tienen las conclusiones generales, donde se reflexiona sobre los resultados obtenidos, y se identifican temas para posibles trabajos futuros.

Capítulo 2

Antecedentes

En este capítulo se describe la teoría básica detrás de los algoritmos genéticos y el análisis de audio empleados en esta tesis. Además se incluye una breve revisión de las investigaciones más relevantes donde se aplican algoritmos genéticos a señales de audio música. Este contenido no es exhaustivo y sólo comprende una introducción a los temas, mostrando lo necesario para familiarizarse con conceptos utilizados en los siguientes capítulos.

2.1. Cómputo Evolutivo y Algoritmos Genéticos

Los algoritmos genéticos pertenecen a una clase de algoritmos conocidos como cómputo evolutivo o programas evolutivos, y están inspirados en procesos biológicos como la selección natural de Darwin. En general, son utilizados para realizar búsquedas y optimizaciones, dando mejores resultados que algoritmos de búsqueda aleatoria cuando se aplican sobre problemas difíciles [McDermott (2008)].

El uso de estas técnicas basadas en la evolución para resolver problemas de cómputo o ingeniería no parece evidente, sin embargo, es claro que la madre naturaleza y todas los seres vivos se enfrentan día a día a problemas complejos en la batalla por la supervivencia. Muchos de estos problemas computacionales requieren buscar a lo largo de espacios grandes con una enorme cantidad de posibles soluciones. Por ejemplo, todos los de movimientos para una partida de ajedrez, las combinaciones de amino ácidos para generar proteínas con determinadas propiedades, o las variaciones de los mercados financieros y bolsas de valores. Buscar y evaluar cada solución válida a lo largo de estos espacios es costoso y poco práctico, por lo que se necesitan algoritmos que sean capaces tanto de hacer búsquedas paralelas así como de utilizar estrategias para disminuir el tiempo perdido evaluando soluciones poco prometedoras [Mitchell (1996)].

Los programas evolutivos poseen estas dos cualidades, ya que pueden operar simultáneamente sobre varias soluciones, además de que con cada iteración hacen menos probable que se evalúen soluciones lejanas a la real. También tienen la ventaja de que pueden adaptarse a cambios en las condiciones iniciales, ya sea que cambie el espacio de búsqueda o las condiciones para evaluar.

Adicionalmente, una de las principales particularidades de estos algoritmos es su relativa simplicidad y facilidad para implementar. No es necesario programar elaboradas reglas para la generación de soluciones, es más, tampoco es necesario conocer a detalle los procesos que dan lugar a esas soluciones. Basta con tener una forma confiable de

evaluar y clasificar el éxito de las soluciones encontradas. Esto puede también hacer que los programas lleguen a resultados innovadores que no hubieran sido obtenidos con métodos tradicionales. Un ejemplo es el diseño de partes mecánicas, como lo son las aspas de ventiladores. No se requiere programar complicadas reglas para generar curvas y superficies que modelen un nuevo diseño. Es suficiente con codificar una evaluación de estos diseños, como puede ser medir el flujo de aire para que un programa evolutivo obtenga buenos resultados.

La evolución biológica presenta todas estas características. En primer lugar las combinaciones de material genético de los seres vivos crean un espacio vasto con gran cantidad de resultados posibles, mientras que las soluciones buscadas son individuos aptos que sean capaz de sobrevivir y pasar sus genes a la siguiente generación. Además, la evolución biológica es altamente innovadora, y ha dado lugar a estructuras complejas que representan una verdadera ventaja competitiva. Tal es el caso del ojo y componentes de visión, donde a partir de células sensibles a la luz, la naturaleza generó sistemas revolucionarios para explorar el mundo con un nuevo sentidos. Para estos procesos naturales los criterios con los que se evalúa el éxito de las soluciones es simple, los individuos viven y se reproducen, o se extinguen. Estas condiciones son constantes sin importar los numerosos cambios que afectan al ambiente, ya sean alteraciones en patrones climáticos, introducción de nuevas especies que compiten directamente, entre otros. Finalmente, las operaciones que rigen este proceso son sencillas: las especies evolucionan mediante variaciones aleatorias como la mutación de genes y la combinación de los mismos mediante la reproducción de individuos.

Existen diversos términos y conceptos que surgen de textos biológicos y son aplicados en todos los algoritmos evolutivos en forma simplificada. Todos los seres vivos contienen ADN (también conocido como *genoma*), el cual es una molécula que codifica toda la información necesaria para construir ese individuo en particular. Este ADN se divide en *cromosomas*, los cuales están compuestos de diversos *genes*. En términos generales, se puede pensar en cada gen como el control de una característica biológica, por ejemplo la estatura, el color de los ojos o la resistencia del sistema immunológico a cierto tipo de enfermedades. Cada gen tiene además un conjunto de valores posibles, llamados *alelos*.

A la colección de genes determinados con ciertos alelos, es decir el genoma con valores específicos, se le conoce como *genotipo*. Con esto se garantiza que dos individuos con el mismo genotipo tendrán las mismas características físicas (siempre y cuando se sometan al mismo ambiente y estímulos). El conjunto de características físicas es conocido como *fenotipo*.

Estos conceptos comprenden toda la información que contiene un individuo de una especie. Al momento en que dos especímenes se reproducen, se genera un nuevo individuo cuyo genotipo depende principalmente de aquel de sus padres. En este proceso ocurre una *combinación* de los genes de los padres, de modo que el nuevo genotipo tiene unos genes del padre y otros de la madre. Adicionalmente, el proceso de combinación no siempre es exacto y se pueden presentar errores al copiar genes. Estos errores introducen genes que no están presentes en ninguno de los padres y se conocen como *mutaciones*. Estas mutaciones son las responsables de introducir nuevos elementos a una especie, y eventualmente pueden llegar a crear especies distintas.

Todos los individuos de una especie forman una *población*. A lo largo de su vida, cada miembro de esta población deberá luchar para sobrevivir, enfrentándose a los retos y desafíos que abundan en el mundo natural. Desde la dificultad en conseguir alimento,

resistir climas extremos y defenderse de depredadores, el mundo está lleno de elementos que hacen una *selección* de cuales individuos de la población vivirán para engendrar la nueva generación. Finalmente, a la probabilidad de que un miembro particular sea seleccionado y sobreviva se le conoce como *fitness*.

Término	Descripción
Genes	Mapeo de un parámetro de la solución.
Cromosoma	Conjunto de genes.
Alelos	Valores posibles para cada gen.
Genotipo	Codificación de todos los parámetros de la solución.
Fenotipo	Visualización de la solución.
Individuo	Posible solución al problema.
Población	Conjunto de todos las soluciones a evaluar.
Mutación	Cambios aleatorios que ocurren sobre cada gen.
Combinación	Intercambio de genes.
Selección	Eliminación de individuos no apto.
Fitness	Evaluación del éxito del individuo.

Tabla 2.1: Términos biológicos y su significado en el cómputo evolutivo.

En el cómputo genético clásico [Mathew (2005)], un cromosoma representa una posible solución al problema. Los genes corresponden a parámetros de algún componente de la solución, por ejemplo las variables de una ecuación. El genotipo normalmente está codificado como una cadena de bits, donde los alelos indican los valores posibles para estos parámetros, determinados por la naturaleza del problema estudiado. La combinación realiza intercambio de genes, tomando secciones de la cadena de bits de dos o más individuos. La mutación realiza cambios aleatorios directamente sobre bits del genoma.

Existen además otras interpretaciones más elaboradas y menos rigurosas, donde los conceptos biológicos no están asignados a bits en todos los casos [Michalewicz (1996), Yu and Gen (2010)]. En estos casos, cada individuo representa una solución candidata, mientras que la población contiene todas las soluciones a evaluar. El genoma de los individuos se compone de genes, los cuales mapean parámetros específicos del problema, sin representarlos como una cadena de bits forzosamente. Los cromosomas son agrupaciones de estos parámetros, que pueden seguir conceptos lógicos del problema, por ejemplo englobar todas las variables de cierto tipo. Mientras que los datos almacenados en el genotipo están codificados de alguna manera, (ya sea en bits, flotantes, o alguna estructura de datos compleja), el fenotipo es la visualización de la solución. Por ejemplo, el genotipo puede almacenar listas de valores reales para las variables x y y, mientras que el fenotipo es la graficación de estas variables siguiendo una función matemática en específico. Los operadores genéticos de combinación, selección y mutación funcionan de manera similar al modelo clásico, sólo que se adaptan para el tipo de codificación empleado en el genotipo. En la tabla 2.1 se muestra un resumen de estos términos empleados en el cómputo evolutivo y su significado general cuando se aplican a problemas de cómputo.

El proceso general de todos los algoritmos basados en evolución se resume en el código 2.1 [Michalewicz (1996)]. Primero se inicializa un población $P(t) = \{x_1^t, x_2^t, ..., x_n^t\}$

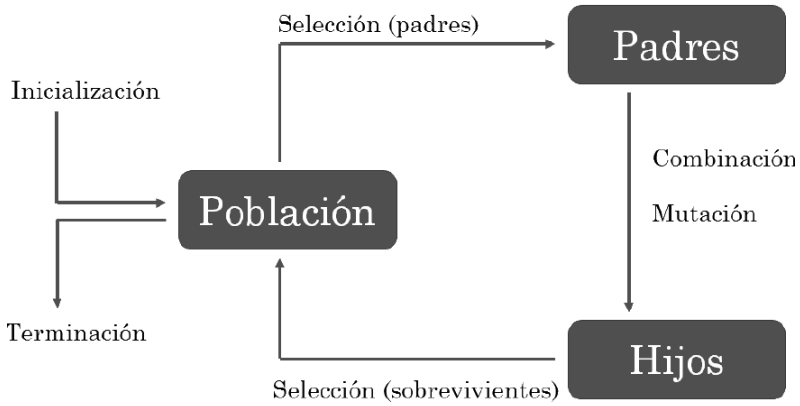

Figura 2.1: Proceso general del algoritmo genético simple.

para el ciclo t y donde n es el tamaño de la población. Cada término x_i^t es un individuo que representa una solución al problema. A continuación se hace una selección de los que individuos x_i^t se van a considerar para generar la siguiente generación. Esta evaluación se hace utilizando algún criterio o función de *fitness*. Estos individuos seleccionados se combinan para generar nuevos términos x_i^t. Algunos de estos individuos pueden tener cambios aleatorios mediante una mutación. Finalmente, se evalúa de nuevo la población $P(t)$ para eliminar los individuos menos aptos y se repite el ciclo (ver figura 2.1).

Código 2.1: Pseudocódigo general para cómputo evolutivo

```
1   inicio
      t = 0
      inicializar P(t)
      evaluar P(t)
      while( $not$ condicion-de-salida) do
6       t = t + 1
        seleccionar P(t) de P(t - 1)
        mutar P(t)
        evaluar P(t)
      fin
11  fin
```

El cómputo evolutivo no siempre ofrece los mejores resultados para todos los tipos de problemas, sin embargo, estos brillan cuando se aplican a problemas difíciles [Yu and Gen (2010)]. Principalmente, son útiles cuando los problemas son multimodales, es decir que existen gran cantidad de mínimos y máximos locales que obligan a algoritmos de búsqueda tradicional a recorrer todo el espacio. También son útiles cuando se manipulan varios parámetros al mismo tiempo, cuando son problemas multivariable. Como un ejemplo sencillo, considérese el problema dado por la ecuación

$$maxf(x) = xsin(10\pi x) + 2$$

$$-1 < x < 2$$

La figura 2.2 muestra la gráfica de la ecuación anterior y se aprecia como hay una gran cantidad de crestas y valles, pero hay un sólo punto donde la función alcanza el valor máximo. Algoritmos tradicionales de búsqueda podrían identificar erróneamente

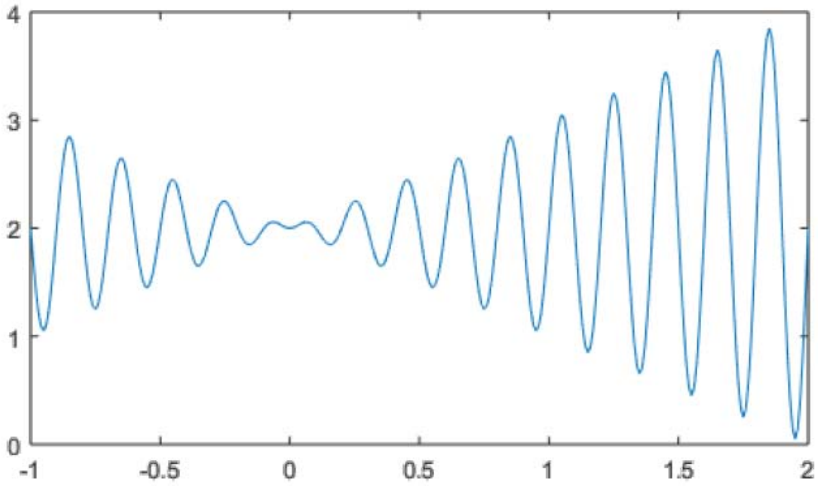

Figura 2.2: Ejemplo de un problema donde el cómputo genético se desempeña bien.

alguna de estas crestas como el valor máximo, y para garantizar que se obtiene el máximo absoluto tendrían que evaluar la función para todos los valores posibles de x. En este tipo de problemas, un algoritmo genético convergería más rápido en la solución real. La población inicial estaría distribuida a lo largo el rango de x, y con cada iteración se irán seleccionando individuos cada vez más cercanos al valor máximo.

Existen diferentes implementaciones dentro de los algoritmos basados en evolución, los más simples siguen el proceso descrito en el código 2.1 con pocas variantes, mientras que los más elaborados agregan operadores complejos, cómo combinaciones a base de ciertas reglas (usualmente relevantes al área de aplicación), o condiciones para seleccionar además de la medida de *fitness*, como por ejemplo un parámetro de longevidad para forzar que los individuos más viejos sean reemplazados después de n iteraciones. Para este proyecto se utilizará el algoritmo genético simple, el cual no utiliza ninguno de estos componentes adicionales.

El algoritmo genético simple se basa en la idea de que las soluciones a problemas complejos se pueden descomponer en "bloques", y que al ir descubriendo, catalogando y combinando estos bloques se llegará eventualmente a la mejor solución Mitchell (1996). Tiene además dos principios básicos. Primero, se obtiene variedad de soluciones a partir de las operaciones de combinación y de mutación. Estos procesos se encargan de explorar todo el espacio de búsqueda. Al ser aleatorios y sin restricciones se evita caer en vicios, por ejemplo se evita permanecer dentro de máximos locales para funciones como la vista en 2.2. Por cuestiones de eficiencia, es importante validar que todo el rango de valores posibles arrojado por estas operaciones genere soluciones válidas. De lo contrario, evaluar estos individuos será un gasto innecesario.

En segundo lugar, este algoritmo obtiene calidad de soluciones a partir de la selección de individuos, todo esto en base a la función de *fitness*. Es crucial que esta función sea confiable y refleje fielmente el comportamiento de las soluciones dentro del problema, ya que este cálculo es lo único que se utiliza para identificar que soluciones son mejores que otras. Y aún cuando se tenga un conjunto de soluciones muy lejanas a la buscada, es necesario poderlas clasificar cuantitativamente. La definición de esta función no es una tarea trivial y depende fuertemente del área del problema a estudiar así como de la naturaleza de los datos de prueba.

2.2. Análisis de sonidos

El término sonido se refiere tanto una sensación auditiva en el oído como a los fenómeno físicos que causan estas sensaciones mediante perturbaciones en algún medio. En la naturaleza, los sonidos enriquecen la información que los seres vivos pueden capturar del ambiente. Con este pueden detectar peligros como depredadores, o en el caso de algunos animales, representar el mundo mediante ecolocación. Aún más importante, el uso de sonidos facilita la comunicación. Desde gruñidos simples que denotan una amenaza hasta complicadas canciones que algunas aves utilizan para encontrar pareja, el sonido es una herramienta vital para la interacción entre individuos y entre especies.

En el caso del ser humano, esta herramienta es aún más compleja, ya que la palabra hablada es quizás la forma más básica del lenguaje. Además existe la música, donde sonidos que individualmente tienen poco valore estético se combinan en forma dirigida y ordenada para comunicar ideas abstractas. Para poder estudiar este campo es necesario familiarizarse con los conceptos básicos del procesamiento de audio digital y las particularidades de la percepción humana del sonido.

2.2.1. Audición y Psicoacústica

El odio humano y el sistema de audición son estructuras complejas, capaces de identificar propiedades clave de sonidos sobre un rango amplio de características físicas. El conjunto del sistema auditivo es sofisticado y lleno de detalles importantes, pero en términos generales se dividen 3 partes: oído externo, medio e interno. En el oído externo el sonido es recolectado y enfocado hacia el oído medio. Contiene elementos como las orejas (*pinna*), el *canal auditivo*, el *tímpano* y en menor medida incluso la cabeza entera afecta al proceso de audición. Además de canalizar y amplificar algunas frecuencias en particular, estos elementos tienen un rol importante para identificar la dirección de sonidos [Young (2007)].

El oído medio funciona principalmente como un acoplador de impedancias, en donde se ajusta la energía acústica del sonido en el aire, en vibraciones acústicas a niveles que pueden manejar los delicados componentes del oído interno. Por otro lado, en el oído interno es donde se analizan e interpretan los sonidos. La *cóclea* (también conocida como *caracol*) es donde se transforma la energía mecánica acústica en impulsos eléctricos que puede interpretar el cerebro. Para esto existe una gran cantidad de células ciliadas que reaccionan a sonidos en diferentes frecuencias, de modo que este sistema opera como un analizador de frecuencia en tiempo real [Everest (1987b)].

El rango de sonidos que puede percibir el ser humano es vasto. La proporción entre los sonidos apenas audibles y aquellos que causan dolor está por el orden de 10^{12}. En cuanto a la variedad de frecuencias, el rango es superior a otros sentidos. La visión sólo distingue entre $4 \cdot 10^{14}$ y $7 \cdot 10^{14}$ Hz, lo que corresponde a poco menos de 1 octava [1]. Mientras que el oído percibe más de 9 octavas [Young (2007)]. Normalmente, el rango oscila desde los 20 Hz hasta casi los 20 000 Hz, pero esto depende de la edad y factores

[1] Una octava se define como el doble de una frecuencia determinada. Por ejemplo, para una frecuencia base de 100 Hz, 1 octava corresponde a 200 Hz, 2 octavas a 400 Hz y así sucesivamente [Everest (1987a)].

ambientales. Lamentablemente al envejecer se pierde la capacidad de escuchar altas frecuencias por lo que el rango disminuye un poco.

Hay 3 características principales del sonido que son relevantes para la audición: tono, timbre, y sonoridad . Estas características están construidas a partir de propiedades físicas, aunque la percepción auditiva no es lineal [Moore (2007)]. La frecuencia fundamental (que es la frecuencia base de todos los sonidos) determina el *tono*. La frecuencia es entonces el reflejo de la altura de los sonidos, lo que da lugar a las diferentes notas musicales. También es útil para por ejemplo distinguir el género de voces, donde voces masculinas son normalmente más graves que las femeninas. El oído es sumamente sensible a cambios de tono, no sólo es capaz de distinguir melodías, sino que obtiene información útil de fuentes más discretas. Por ejemplo, es capaz de identificar si una oración hablada es una afirmación o una pregunta.

La amplitud de las ondas de sonido determinan la *sonoridad*, y en cierto modo se puede pensar como el volumen o nivel de un sonido, pero no sigue un comportamiento lineal. En su forma más básica, el nivel de una señal de audio se mide en dB, donde un sonido a 1 dB corresponde al mínimo que puede percibir el oído. Sin embargo, el oído no es igualmente sensible dentro de todo el rango de frecuencia, por lo que es más fácil detectar sonidos alrededor de los 3 kHz que en los extremos del espectro. Este comportamiento es conocido y se ha representando mediante *curvas isofónicas*. Éstas se muestran en la figura 2.3 y miden con precisión la respuesta de la audición. Con estas curvas es posible identificar que un tono puro con frecuencia de 1 kHz y una intensidad de 20 dB, tendrá la misma sonoridad aparente que un tono puro de 20 Hz a casi 90 dB. No obstante, es importante notar que la diferencia no es tan grande a niveles mayores, donde cerca de los 90 dB se estabiliza para la mayor parte del rango auditivo [Everest (1987b)]. Adicionalmente, la percepción de sonoridad depende del tipo de sonidos, donde sonidos con un mayor ancho de banda son más sonoros.

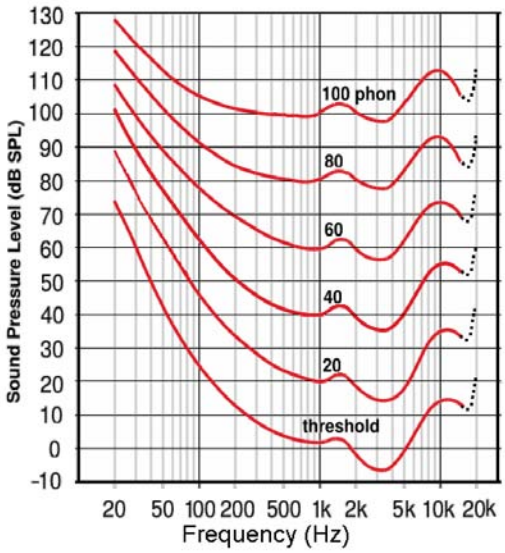

Figura 2.3: Curvas isofónicas de igual sonoridad para tonos puros. Tal como se muestra en el estándar ISO 226:2003.

Cada sonido tiene además un cierto "color", atributo conocido como *timbre*. Este timbre contiene la información que permite diferenciar el origen entre dos sonidos con el mismo tono y nivel. Es posible distinguir entre dos instrumentos musicales ejecutando

la misma nota, o incluso identificar personas individuales entre un grupo recitando las mismas palabras. El timbre es un propiedad compleja que depende de muchos factores, pero principalmente se basa en la forma de onda de una señal sonora. Esta forma de onda, determina a su vez el contenido espectral de la señal. Para un tono puro generado a partir de una onda senoidal, se tendrá en el espectro una sola componente en la frecuencia de dicha senoidal. Para sonidos complejos, estarán presentes componentes adicionales, conocidas como parciales. Algunas de estas parciales caerán dentro de múltiplos enteros de la frecuencia fundamental, y se conocen como *armónicas*. Por ejemplo, si se tiene un sonido con frecuencia fundamental de 215 Hz, se tendrán energía en el espectro a 215, 430, 645, ... $215n$, donde n es el número de armónico. Algunos sonidos no siguen este patrón y tienen un espectro aparentemente aleatorio. La distribución y nivel de estas parciales son la principal contribución al timbre de señales, y una pequeña clasificación se muestra en la tabla 2.2.

Espectro Gral.	Fuente	Instrumentos
Todas armónicas parciales	Columnas de aire abiertas, vibración de cuerdas	Metales, guitarra, bajo, alientos madera
Armónicas parciales impares	Columnas de aire cerradas	Clarinete, órgano, silbatos
Parciales NO armónicas	Membranas, barras suspendidas	Piano elétrico, platillos, vibráfonos
Parciales aleatorias	Transientes, percusiones	Truenos, respiracion, batería

Tabla 2.2: Clasificación de sonidos según el contenido de armónicas y ejemplos de instrumentos musicales.

El espectro del sonido a su vez puede variar en el tiempo, de modo que las diferentes armónicas presentes pueden no tener la misma duración, o incluso pueden tener variaciones en amplitud. El timbre es además multidimensional, ya que al depender de varios factores no se puede clasificar dentro de una escala simple. Algunos de estos factores son 1) si es periódico 2) las fluctuaciones del espectro 3) desarrollo en el tiempo de la señal en general 4) modulaciones sobre alguno de esos factores. Esta dependencia del tiempo quedó demostrada en varios experimentos donde sonidos fueron clasificados como instrumentos diferentes al ser escuchados después de ser procesados en reversa [Moore (2007)]. Otro ejemplo es la flauta, que a pesar de que el espectro tiene una combinación relativamente simple de armónicas parciales, cada nota es acompañada por un poco de ruido producto del flujo de aire repentino.

Otras características son menos importantes, tal es caso de la fase. Algunos experimentos han demostrado que la percepción de sonidos no se afecta por cambios de fase, cuando ocurren dentro de ciertos límites. Principalmente, el oído no es sensible a la fase excepto cuando estos introducen distorsiones a la forma de onda [Everest (1987b)].

2.2.2. Sonidos Cortos

El tipo de señales que se analizaran en este trabajo corresponden a sonidos cortos. Las fuentes están enfocadas a instrumentos musicales, o a sonidos que sean relevantes para generar música. Las señales se componen de un sólo instrumento ejecutando una o más notas. La duración de las señales es corta, pero suficientemente larga para contener el desarrollo natural del sonido. Este desarrollo incluye secciones donde el nivel del sonido aumenta, donde se mantiene y donde disminuye para finalmente desaparecer. También contiene la modulación natural del espectro de frecuencia.

Un par de ejemplos se muestran en la figura 2.4, donde se tiene un sonido generado por piano y otro por guitarra eléctrica. Al ver las señales en el tiempo no es fácil identificar información relevante, salvo la complejidad de la mismas. En los espectrogramas se aprecia que ambas tienen gran cantidad de información. No sólo hay parciales armónicas sino que se observan regiones con gran energía a lo largo de todo el espectro. La señal de piano es un poco más estable, ya que tiene menor variación en el tiempo. Por otro lado, la señal de guitarra eléctrica muestra una caída rápida de la energía en las frecuencias superiores después de un breve periodo inicial. A pesar de que musicalmente son sonidos sencillos, es evidente que el audio es complejo.

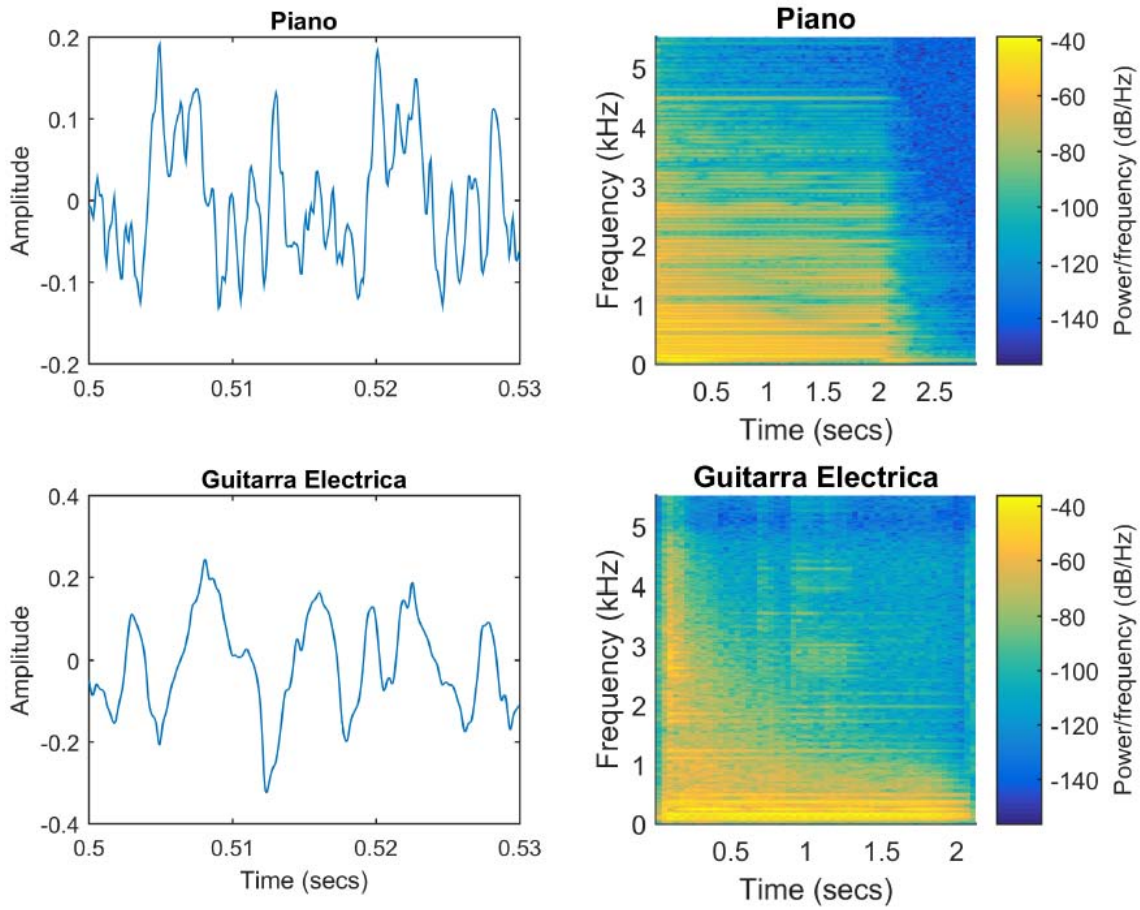

Figura 2.4: Ejemplos de sonidos cortos. Las imágenes superiores muestran una nota de piano vista en el tiempo y el espectrograma resultante. Las imágenes inferiores meustran el mismo caso para un acorde ejecutado por guitarra eléctrica con distorsión.

2.3. Estado del arte

Se ha escrito una gran cantidad de artículos y reportes donde se explora la aplicación de cómputo evolutivo a numerosos problemas en diversos campos. Sin embargo, la aplicación a señales musicales y de audio continua poco explorada. Uno de los experimentos comunes son la aproximación de parámetros de un sintetizador en específico. Otros experimentos se enfocan en el contenido musical de señales, buscando métodos para transcripción o composición automática.

Como trabajo introductorio está el capítulo escrito por Husbands et al. (2007), donde se incluye un breve resumen del cómputo genético y una explicación detallada del algoritmo genético simple. Esta explicación se enfoca en la definición original y utiliza un representación del genotipo como una cadena de bits. Posteriormente, el artículo habla sobre aplicaciones concretas de este algoritmo a temas relacionados con música. Primero habla de composición de temas musicales, donde se utiliza el algoritmo genético para generar las melodías que crean piezas musicales para diversos géneros. Cabe resaltar que para este caso, no se hace uso de análisis de audio, ya que se consideran las notas musicales directamente, y todas las reglas y restricciones operan sobre estas notas y no sobre sonidos. Este acercamiento es interesante pero no es relevante para los temas vistos en esta tesis.

El mismo escrito también menciona aplicaciones de algoritmos genéticos a diseño sonoro, donde ya se aplican directamente sobre audio. Menciona dos corrientes principales, una para sintetizar sonidos y otra para procesar sonidos existentes mediante efectos. Para el primer caso, describe algunos experimentos donde se utiliza el algoritmo genético para optimizar los parámetros de un sintetizador en específico para obtener una forma de onda deseada, cuyo mayor problema es la definición de una función de *fitness* ideal. El artículo no profundiza en los detalles de la implementación de estos experimentos, sólo da una introducción y menciona los puntos más relevantes.

El artículo por Macret et al. (2012) se enfoca en uno de los puntos mencionados en el trabajo anterior. Aquí se explora el uso de algoritmos genéticos para optimizar los parámetros de un sintetizador y poder igualar sonidos. La síntesis utilizada es la síntesis FM modificada, la cual es un proceso complejo y costoso, pero que genera sonidos que son muy apreciados por la comunidad musical. Los sonidos objetivo utilizados corresponden a instrumentos musicales que tienen armónicas parciales claras. La función de *fitness* empleada obtiene las principales parciales mediante el análisis de espectrogramas y descarta lo demás. Los resultados son buenos y se demuestra que el algoritmo genético es útil para aproximar los parámetros.

El trabajo publicado por Manzolli et al. (2001) contiene un acercamiento diferente al mismo problema. Aquí también se utiliza un algoritmo genético para aproximar parámetros de un sintetizador, sin embargo, la función de *fitness* empleada es interactiva, es decir que tiene una retroalimentación de un usuario. El sintetizador empleado es un software conocido como *ESSynth* y se basa en la manipulación directa de formas de onda para generar sonidos. El rango de sonidos que puede generar este sintetizador es limitado, pero los resultados muestran que el algoritmo genético funciona para replicar sonidos. Las limitantes de este sistema es que los sonidos objetivo todos fueron generados previamente con el mismo sintetizador y que la función de *fitness* interactiva es lenta y por tanto no es posible realizar ejecuciones largas.

Finalmente, en la tesis de McDermott (2008) se expanden los temas anteriores.

Realiza experimentos para usar algoritmos genéticos con funciones de *fitness* tanto interactivas como tradicionales. Además, define una variedad de atributos y propiedades para describir las señales de audio empleadas, incluyendo varios enfocados a la percepción de timbre y comparaciones entre estos. Este trabajo contiene una gran cantidad de datos donde demuestra el desempeño de los algoritmos genéticos para la replicación de audio, además de que define métricas y puntos de control. El rango de señales utilizado es limitado.

Por otro lado, en el siguiente capítulo se hace un estudio a detallado de una implementación de algoritmos genéticos aplicado a la replicación de imágenes, donde se ejemplifican varios conceptos.

Capítulo 3

Caso de Estudio EvoLisa

3.1. Introducción

Para el desarrollo de los experimentos de esta tesis, se utilizó como ejemplo y como inspiración un software ya existente. Este software se conoce como EvoLisa y fue construido por Roger Johanson y utiliza algoritmos genéticos para replicar imágenes, combinando polígonos simples [Johanson (2015)].

La teoría detrás de este proyecto es que una imagen cualquiera puede ser representada por una combinación de n polígonos regulares de un solo color. Por lo tanto, una imagen está compuesta de una colección de polígonos. Cada polígono está compuesto de un color más una colección de puntos. Cada color está compuesto por 4 componentes (canal rojo, canal verde, canal azul y canal alfa). Y cada punto está compuesto por una posición en x y una posición en y.

3.1.1. Principales Características

Carga de imágenes El programa puede leer imágenes en diferentes formatos y despliega una visualización de éstas. No tiene restricciones en cuanto al tamaño o resolución de las imágenes pero se debe considerar que mientras más pixeles tenga la imagen más tiempo se tardará el proceso.

Tamaño de resultado El programa permite modificar el tamaño de la imagen resultado para poder apreciar mejor el resultado. Esta modificación es sólo para fines ilustrativos y no afecta ni a la imagen original ni al resultado.

Interfaz de usuario responsiva El proceso de evolución opera en un hilo independiente a la interfaz de usuario por lo que ésta no se bloquea y se actualizan constantemente indicadores sobre el estatus de operación del algoritmo.

Guardar o Cargar DNA Al obtenerse una imagen resultado, es posible serializar y guardar el DNA del resultado a un archivo. Este archivo puede ser compartido con otros usuarios o ser utilizado como base para comenzar un nuevo proceso de evolución.

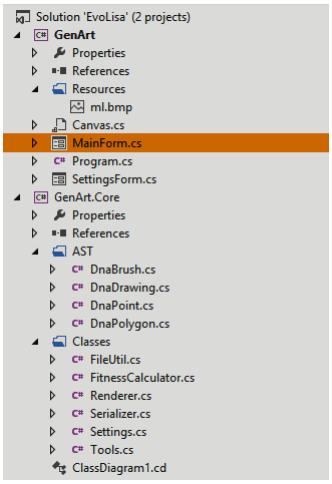

Figura 3.1: Estructura de la solución para EvoLisa.

Opciones Los parámetros utilizados para determinar las probabilidades de mutación, así como el rango de mutación se determinan antes de iniciar la evolución. La gran cantidad de parámetros ofrece una alta flexiblidad al momento de ejecutar evoluciones.

Algoritmo genético simple Los implementación del algoritmo genético es simple y no tiene variaciones, por lo que es fácil de estudiar.

3.2. Estructura del programa

El proyecto está construido en C# y .net. La solución está dividida en 2 proyectos: *GenArt* y *GenArt.Core* (ver figura 3.1). El primer proyecto contiene la parte visual del programa, incluyendo la interfaz de usuario y recursos necesarios como imágenes, mientras que en el segundo proyecto se definen las clases que componen los elementos a evolucionar así como clases auxiliares, por ejemplo un serializador del ADN de un objeto solución. La ejecución del algoritmo genético se encuentra implementada directamente en el primer proyecto, dentro del código pantalla principal del programa.

3.2.1. Principales Clases

En el proyecto *GenArt.Core* se tienen las clases que componen el objeto que representa la solución al problema (ver figura 3.2), es decir una imagen resultado. Estas clases se utilizan tanto para guardar la información de una solución individual como para realizar las operaciones de mutación que utiliza el algoritmo genético.

La clase principal es *DnaDrawing*. Como propiedades únicamente tiene un contador de la suma total de puntos de todos los polígonos que contiene esta imagen, y una colección de objetos *DnaPolygon* que representan los polígonos que conforman la imagen resultante. Cada uno de estos polígonos tiene a su vez un color definido por el objeto *DNABrush* y una colección de puntos dados por los objectos *DNAPoint*. Finalmente, estos puntos contienen solamente valores para la posición en x y en y.

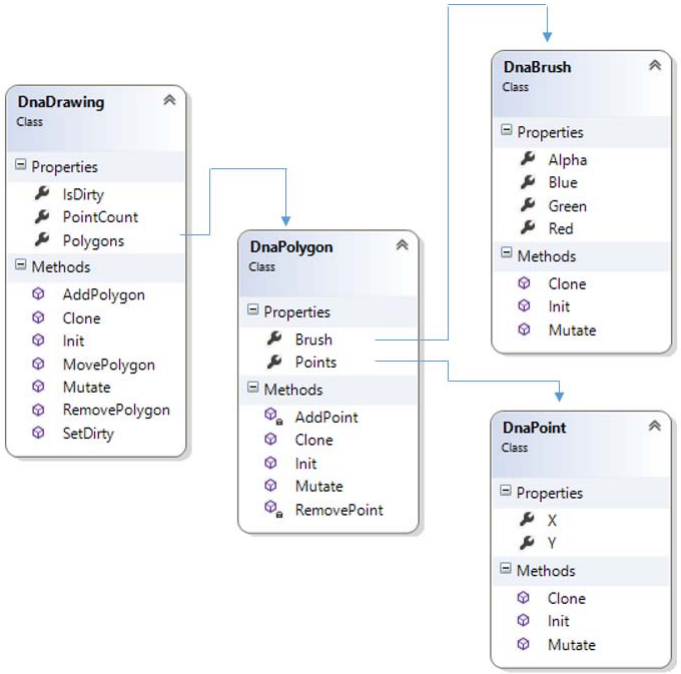

Figura 3.2: Diagrama de clases para representar una solución.

En cuanto a métodos, cabe señalar que cada objeto define una función para mutar, dentro de la cual se pueden realizar diferentes acciones. Para el caso de *DNAPoint* la mutación implica un cambio en las coordenadas de posición x y y, sin embargo, existen 3 tipos de posibles de modificaciones donde varía que tanto se puede desplazar un punto (despalazamineto cercano, mediano y completamente aleatorio). La modificación que se utilizará es determinado por los parámetros de evolución (definidos desde la pantalla principal), donde se asigna una probabilidad a cada tipo de movimiento. Además se valida que la nueva posición de los puntos esté dentro de los límites de la imagen.

Para el caso de *DNABrush*, la mutación incluye un cambio para cada componente (RGBA) del color y cada canal puede cambiar en una proporción diferente. Al igual que en *DNAPoint*, hay diferentes tipos de movimientos que modifican el rango disponible para este cambio. Para *DnaPolygon* la mutación opera de manera ligeramente diferente. Primero se ejecutan funciones para remover o agregar puntos a la colección de *DNAPolygon* y después se invoca la mutación del color *DNABrush* asociado y la mutación de cada punto *DNAPolygon*.

Por último para el objeto *DnaDrawing* la mutación consiste de operaciones para agregar, remover o cambiar el orden de los polígonos *DNAPolygon* y después se ejecuta la mutación de cada polígono. Aquí cabe resaltar que el orden en que se dibujan los polígonos es relevante, ya que como los colores contienen canal alfa, es posible tener colores con transparencias, y la imagen que resultará de sumar estos colores dependerá del orden en que se vayan dibujando los polígonos cuando éstos se sobreponen.

3.2.2. Implementación Algoritmo Genético

El algoritmo genético en sí se ejecuta directamente en el código de la pantalla principal y es bastante simple. Primero inicializa un objeto *DnaDrawing* que va a representar la solución óptima y la imagen que se va a mostrar como resultado. Como característica particular de este algoritmo, la inicialización aplica un color negro a toda la imagen, por lo que es más sencillo replicar imágenes obscuras que claras. Sin embargo, se podría modificar fácilmente esta inicialización para adaptarse a otro de imágenes, usando incluso fondo transparente.

Después de inicializar comienza el ciclo del algoritmo genético. Este es un ciclo *while* delimitado por una bandera que se fija desde la pantalla. De modo que el ciclo continuará hasta que se cancela la evolución desde la interfaz de usuario y no existe otra condición de salida. El ciclo consiste en crear un clon del objeto *DnaDrawing* actual, y mutarlo (lo que invocará todas las mutaciones de los objetos que lo componen). En caso de que alguna de las mutaciones de este nuevo objeto haya realizado cambios al *DNA*, entonces se calcula la *fitness* del mismo y se hace la comparación entre la *fitness* de la solución óptima actual con la del objeto nuevo. Finalmente se asigna como solución óptima actual el objeto con mejor *fitness* y el individuo no seleccionado es descartado.

Adicionalmente se incrementan variables que sirven de indicadores del proceso de evolución, como lo son la cantidad de generaciones y de objetos seleccionados. También cabe mencionar que el algoritmo utiliza candados (*lock*) para garantizar la operación multi hilo del algoritmo y mantener la interfaz de usuario activa.

Código 3.1: Función que implementa el algoritmo genético en *EvoLisa*

```
private void StartEvolution(){
   SetupSourceColorMatrix();
   if (currentDrawing == null)
       currentDrawing = GetNewInitializedDrawing();
   lastSelected = 0;

   while (isRunning)
   {
       DnaDrawing newDrawing;
       lock (currentDrawing)
       {
        newDrawing = currentDrawing.Clone();
       }
       newDrawing.Mutate();

       if (newDrawing.IsDirty)
       {
        generation++;

        double newErrorLevel = FitnessCalculator.GetDrawingFitness(newDrawing, sourceColors);

        if (newErrorLevel <= errorLevel)
        {
         selected++;
         lock (currentDrawing)
         {
             currentDrawing = newDrawing;
         }
         errorLevel = newErrorLevel;
        }
       }
   }  //else, discard new drawing
}
```

3.2.3. Interfaz de Usuario

La interfaz de usuario del programa es minimalista y funcional (ver figura 3.3). Incluye elementos para visualizar las imágenes de origen y el mejor resultado actual de la evolución, controles para ejecutar y detener la operación y algunas etiquetas que muestran indicadores claves del algoritmo. Además hay una pantalla adicional donde se especifican las probabilidades de mutación y los límites de desplazamiento para los valores mutables.

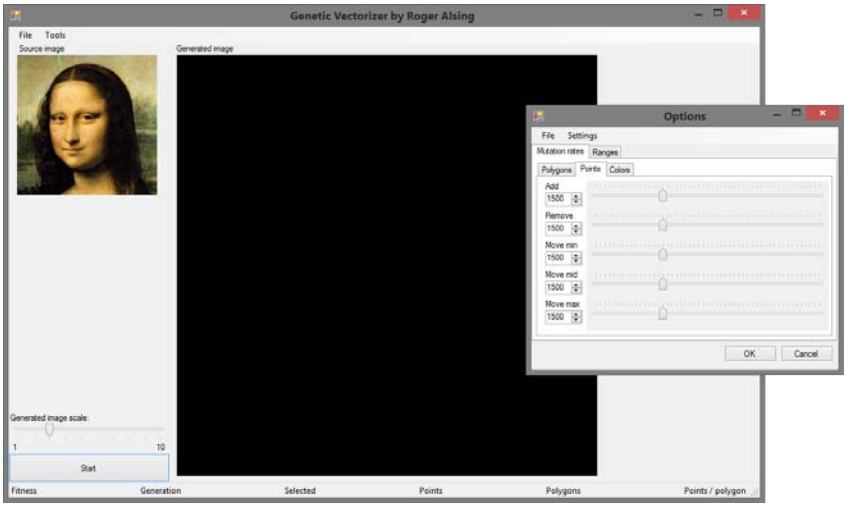

Figura 3.3: Interfaz de Usuario de EvoLisa.

Una de las características más importantes de la interfaz de usuario es que el código se ejecuta en un hilo adicional, por lo que la pantalla continua operable mientras se ejecuta el proceso. Esto permite ir visualizando en tiempo real la imagen del mejor resultado actual y ver como van evolucionando los resultados. Este detalle es muy útil para apreciar la influencia de los parámetros evolutivos en la ejecución del algoritmo de forma rápida y clara, lo que facilita la experimentación y comprensión de los efectos del algoritmo genético.

3.3. Parámetros de Algoritmos Genéticos

Los parámetros del algoritmo genético del programa son determinados desde la construcción y a excepción de las probabilidades de mutación para cada clase, no pueden ser modificados. El programa utiliza una implementación de algoritmos genéticos y no explora operadores complejos.

3.3.1. Población

El programa utiliza una población de sólo dos individuos, que consiste de un individuo y su clon mutado. De esta manera cada individuo compite contra si mismo y sólo las mutaciones introducen variedad a la población. El individuo inicial consiste en una imagen vacía, sin polígonos, y se inicializa llenando de color negro todo el fondo. Conforme avanza el algoritmo, cada generación introduce un individuo adicional que

es una copia del individuo original y se le aplican mutaciones (sujetas a las condiciones normales de mutación), sin embargo sólo se conserva el individuo con mejor *fitness*, por lo que sólo las mutaciones que mejoran el resultado se conservan.

De este modo, todas las evoluciones comienza a partir del mismo individuo, pero es posible comenzar a partir de un individuo evolucionado previamente si se guardó el *DNA* correspondiente. En este caso, la población inicial cambia.

3.3.2. Mutación

Como se describió anteriormente el programa implementa diversas mutaciones distribuidas a lo largo de todos los objetos que componen una solución. A diferencia de lo que indica la teoría clásica de algoritmos genéticos donde el *DNA* del objeto debe representarse como un arreglo de bits y la mutación opera sólo sobre este arreglo, la implementación de EvoLisa utiliza un acercamiento más pragmático. No existe una operación única de mutación que opere sobre el *DNA* completo, sino que existen numerosas mutaciones diferentes y cada una opera sobre un elemento particular de lo que compone el genotipo del individuo. Este acercamiento consigue evitar el proceso de representar el *DNA* del individuo en un arreglo y viceversa, además de que es una aplicación ingeniosa del paradigma orientado a objetos.

Asimismo, otra particularidad del programa es que algunas de las mutaciones pueden agregar o remover objetos. Por ejemplo, un polígono *DNAPolygon* contiene una colección de puntos que determinan la forma del polígono, pero el tamaño de está colección no es fijo. Mediante mutaciones, se pueden ir agregando o removiendo puntos lo que cambia la forma del polígono. Este tipo de operaciones no serían posibles con una implementación de algoritmos genéticos donde el tamaño del *DNA* es fijo.

Otra de las ventajas de este diseño , es que se pueden asignar rangos y probabilidades a cada operación de mutación diferente. Esto ofrece gran control sobre el proceso de evolución, ya que es posible favorecer algunas operaciones. Los valores por defecto de estos parámetros de mutación hacen que los individuos tiendan a eliminar elementos del *DNA*, lo que disminuye la complejidad y tamaño del genotipo de los individuos.

3.3.3. *Fitness*

La función para evaluar los resultados implementada por el programa compara cada pixel del resultado con el pixel correspondiente de la imagen original y obtiene la diferencia del valor de los canales RGB. La diferencia total de cada pixel se determina con la suma del cuadrado de las diferencias de cada canal (ver código 3.2). Por último, el valor total de *fitness* del resultado se determina sumando los valores de diferencias de todos los pixeles.

De este modo, el valor de *fitness* más bien corresponde a una medida de error, de que tan diferente es el resultado a la imagen original. Donde una copia perfecta tendría un valor de *fitness* igual a 0. No obstante, para valores diferentes a 0, esta medida ofrece poca información sobre la percepción humana de las imágenes, y solo es útil para realizar comparaciones.

Código 3.2: Cálculo de *fitness* por cada pixel

```
private static double GetColorFitness(Color c1, Color c2)
```

```
2  {
       double r = c1.R - c2.R;
       double g = c1.G - c2.G;
       double b = c1.B - c2.B;

7      return r*r + g*g + b*b;
   }
```

El formato de color RGB a 24 bis utilizado por el programa determina que cada canal tiene un valor entre 0 y 255. La diferencia máxima para cada canal es por lo tanto de 255, y el valor de error para cada pixel está determinado por la ecuación:

$$Error(x, y) = (x_r - x_r)^2 + (x_g - y_g)^2 + (x_b - y_b)^2$$

donde x y y son pixeles individuales de cada imagen.

Es importante mencionar que este algoritmo no considera el canal alfa. La influencia del canal alfa se ve reflejada en la forma en como se construyen las imágenes. Al ser una combinación de polígonos que van sobreponiendo, cuando el color de algún pixel tiene transparencias el color resultante del pixel será influenciado por éstas. Por lo que no es necesario tomar en cuenta al canal alfa de manera aislada.

3.4. Resultados

Se ejecutaron 4 experimentos cortos para evaluar la operación del programa. En cada experimento se cargó una imagen específica y se ejecutó el algoritmo para obtener un resultado. Para poder apreciar mejor los resultados se utilizó la herramienta del programa para expandir la imagen resultado en un orden de 6 veces. Las imágenes de comparación mostradas en este documento aplican la misma expansión a las imágenes base, lo que introduce imperfecciones que no fueron consideradas por el algoritmo. Cada experimento se ejecutó el tiempo necesario para obtener aproximadamente 50 000 generaciones, registrando los datos de la evolución y las imágenes resultado. Para el último experimento, la evolución se dejó correr por 25 minutos, para obtener un resultado más detallado.

3.4.1. Primer Experimento

Para el primer experimento se utilizó una imagen muy simple a modo de control. La imagen consiste en un fondo negro cubriendo la mayoría del canvas y se tiene una figura triangular de un solo color, sin embargo, al no ser un triangulo perfectamente geométrico, el borde de esta figura no es preciso y tiene ligeros accidentes que lo desvían de una línea recta.

Se operó el algoritmo genético por poco mas de 50 mil generaciones lo que tomó alrededor de 3 minutos. La imagen resultante es buena y tiene un alto parecido a la imagen original. El contorno triangular es muy bueno y los colores son casi perfectos. No obstante, se observan unas lineas adicionales que salen de la figura principal que no están presentes en la imagen base.

Analizando los indicadores (tabla 3.1) se observa que a pesar de la que la imagen resultado se parece bastante a la original, la *fitness* final es un valor grande, marcadamente diferente de 0. Y a pesar de que la imagen original es muy simple, se utilizaron

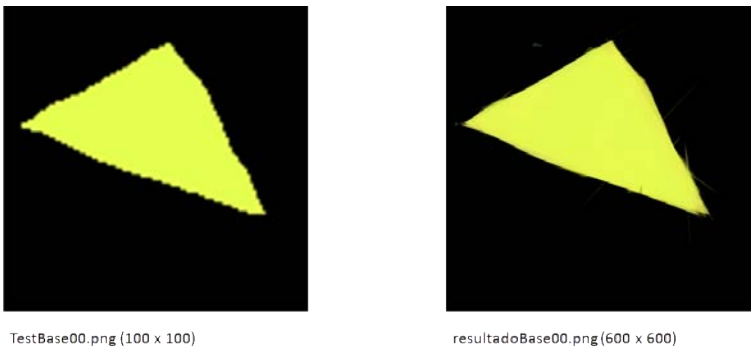

Figura 3.4: Primer experimento EvoLisa.

32 polígonos, con 8 puntos por polígono en promedio. Este caso además tiene la ventaja de que el fondo es negro, y es el mismo color con el que se inicializan las imágenes en el algoritmo (ver figura 3.4).

3.4.2. Segundo Experimento

El segundo experimento consiste de una imagen un poco más elaborada pero compuesta de elementos muy básicos. Utiliza un fondo de color claro y tres figuras geométricas simples de color sólido donde algunas se sobreponen. El color claro del fondo representa un reto considerable para el algoritmo ya que éste primero llena todo el canvas con color negro, y como esta imagen tiene un fondo claro se va a necesitar invertir proceso para obtener el fondo.

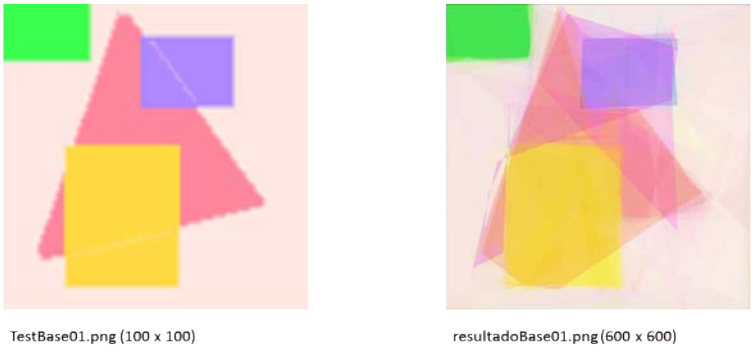

Figura 3.5: Segundo experimento EvoLisa.

Se operó el algoritmo genético de manera similar al experimento anterior, obteniendo alrededor de 50 mil generaciones y un tiempo de ejecución de 3 minutos. El resultado es bastante bueno pero se aprecian considerables deficiencias. El color del fondo es prácticamente idéntico y se muestra claramente las regiones y colores de las 4 figuras geométricas. Sin embargo, las zonas donde las figuras se sobreponen presentan imperfecciones, e incluso se observan que hay polígonos que forman parte de dos o más figuras, y polígonos que cubren zonas donde solo hay fondo.

De los indicadores (tabla 3.1) se observan que en general se tienen valores muy parecidos al experimento pasado, pero que la *fitness* es mucho mayor en este caso, lo que refleja que este resultado tiene mayor cantidad de error o diferencia con la imagen

base. Además se tiene mayor cantidad de polígonos pero la cantidad promedio de puntos por polígonos es prácticamente la misma (ver figura 3.4).

3.4.3. Tercer Experimento

El tercer experimento utiliza una imagen más compleja que ya es una buena aproximación a lo que podría ser una imagen común y corriente. La imagen consiste de un fondo multicolor, con zonas de forma irregular que no son fáciles de representar con polígonos, además de una figura central sobrepuesta en forma de Z de color blanco.

La ejecución del algoritmo fue similar a los casos anteriores, obteniendo alrededor de 50 mil generaciones durante 3 minutos. El resultado es significativamente peor a los anteriores, no obstante todos los elementos principales de la imagen son apreciables y en algunos casos bastante cercanos a la original. El fondo fue lo que más sufrió, el programa pudo identificar correctamente las 7 zonas diferentes y los colores son muy acercados, pero los bordes entre las zonas, algunos de los cuales consisten de curvas suaves y regiones pequeñas son malos. La figura central en forma de Z fue plasmada de manera precisa, tanto en forma como color. Y en general, a lo largo de todo el resultado se puede apreciar ruido en las partes donde se sobreponen polígonos.

Figura 3.6: Tercer experimento EvoLisa.

Los indicadores (tabla 3.1) presentan valores similares al experimento anterior, salvo que la *fitness* es un orden de magnitud mayor, lo que acertadamente mide la similitud del resultado con la imagen objetivo. Cabe señalar que la cantidad de polígonos y puntos fueron muy parecidos en este experimento y el anterior (ver figura 3.6).

3.4.4. Cuarto Experimento

Para el último experimento, se usó la imagen que da nombre al programa, una recorte de la Monalisa de $200x200$ pixeles. A diferencia de las imágenes anteriores, ésta es altamente compleja, con gran cantidad de detalles tanto en formas como colores. Y aunque la paleta de colores tiende a dos zonas principales (obscuros para el cabello y fondo y claros para la piel) hay una gran variedad de tonos y no son colores sólidos.

En este caso, para considerar la mayor complejidad de la imagen, se ejecutó el algoritmo por mayor tiempo, llegando a las 150 mil generaciones en 25 minutos. El resultado es en general bueno pero se pierden los detalles finos. La forma es general es muy buena, se identificó claramente el contorno del cabello, rostro y cuello, e incluso

Figura 3.7: Cuarto experimento EvoLisa.

se alcanzan a distinguir detalles más sutiles como los rasgos faciales y ligeras sombras sobre la boca y cuello. Los colores también son buenos ubicando claramente los que corresponde a la cabeza y al fondo. Por otro lado, a pesar de que el rostro presenta detalles suaves, el fondo es muy simple y consiste de grandes bloques de color sólido.

Los indicadores (tabla 3.1) muestran gran parecido al experimento anterior, a pesar de que el tiempo de ejecución fue mucho mayor. La *fitness* es muy similar a la imagen anterior, y aunque se tiene considerablemente más polígonos y puntos en este experimento, la cantidad promedio de puntos por polígonos es la misma (ver figura 3.7).

3.4.5. Comparación de resultados

En la tabla 3.1 se muestra la comparación de los indicadores de los diferentes experimentos. Se aprecia que para el segundo experimento, la cantidad de individuos seleccionados fue mayor a la del primer y segundo experimentos, lo que presume una evolución más paulatina con menores incrementos de *fitness* por selección. En cuanto a la cantidad de polígonos y puntos generados, se observa que los experimentos 2 y 3 tienen un número muy similar, a pesar de ser ampliamente diferentes. Esto es congruente con la complejidad de las imágenes, ya que formas más elaboradas como las que están presentes en el experimento 3, requieren de una mayor cantidad de puntos para delinear los contornos. Finalmente, resalta que la cantidad promedio de puntos por polígono es prácticamente constante para todos los experimentos, y se aproxima al máximo permitido por el programa.

Exp	Fitness	Gen.	Selecc.	Puntos	Polígonos	Puntos x Polígono	Tiempo
1	982 462	50 293	4 850	255	32	7	197
2	4 342 990	50 246	7 810	424	51	8	226
3	24 318 263	52 584	6 500	472	55	8	206
4	22 570 840	158 511	10 884	723	84	8	1 500

Tabla 3.1: Indicadores de experimentos de EvoLisa.

Capítulo 4

Diseño e Implementación - Generación de Audio con Algoritmos Genéticos

4.1. Introducción

En este capítulo se describe el diseño de los experimentos así como la composición de los principales elementos del programa que se construyó.

4.2. Estructura General del Programa

El programa fue divido en dos secciones. Por un lado se creó una biblioteca de funciones en *MATLAB* que realizan la parte propia del procesamiento de señales. Esto incluye tareas simples como la lectura y guardado de archivos de audio, y actividades más especializadas como el cálculo de espectrogramas, extracción de características, cálculos de distancias entre señales y graficación de resultados. Además se creó una aplicación de escritorio desarrollada en *.Net* y *C#* que contiene la interfaz de usuario y la implementación del algoritmo genético. Las funciones creadas en *MATLAB* son invocadas directamente por esta aplicación.

4.2.1. Principales Características

Carga de sonidos El programa puede leer archivos de audio en diferentes formatos, incluyendo *.mp3* y *.wav*. No tiene restricciones en cuanto a la frecuencia de muestreo o profundidad de bits utilizada, sin embargo, si los archivos son estéreo o contienen más canales, todos serán sumados en un solo canal para realizar el proceso en modo monoaural. Esto se diseñó así para disminuir el tiempo de ejecución además de que para utilizar sonidos estereofónicos correctamente es necesario considerar numerosos fenómenos psicoacústicos que van más allá del alcance de esta tesis.

Interfaz de usuario responsiva Siguiendo el ejemplo de *EvoLisa* la aplicación opera de manera multi hilo, de modo que la interfaz de usuario no se bloquea mientras

se ejecuta la evolución o cuando se hacen llamados a funciones de *MATLAB*. Además la pantalla principal actualiza constantemente diversos indicadores y gráficas.

Sintetizador Manual Se tiene una pantalla adicional que utiliza las funciones de síntesis de audio donde se pueden generar sonidos de manera manual, sin ejecutar el algoritmo genético. Los sonidos generados por esta pantalla pueden ser exportados en formato de audio o pueden ser guardados en formato propio de la aplicación y ser utilizados posteriormente como base para otro sonido o para iniciar una evolución.

Guardar o Cargar DNA Al igual que en *EvoLisa*, el programa permite guardar y cargar el resultado de una evolución, y puede ser utilizado para iniciar una siguiente evolución o para compartirse con otros usuarios. En este caso, el archivo de solución contiene datos adicionales sobre la trayectoria de la evolución que son necesarios para la graficación de resultados.

Opciones y Parámetros La pantalla de opciones tiene apartados para configurar las probabilidades de mutación, los rangos de valores disponibles para las mutaciones y las propiedades del espectrograma y función de distancia que se empleará en el algoritmo. Es importante considerar cuidadosamente estas opciones ya que influyen gravemente sobre los resultados de la evolución. Todos estos parámetros se pueden serializar y guardar, para ser cargados posteriormente, lo que da agilidad al momento de preparar diferentes experimentos.

Visualización de Resultados La pantalla principal contiene acciones para poder visualizar tanto el sonido objetivo como el mejor resultado actual de la evolución. Es posible reproducir los sonidos, consultar el espectrograma, o ver una gráfica personalizada que contiene la señal en el tiempo, el cálculo de la DFT de toda la señal, y un espectrograma reducido.

Graficación de resultados Para poder analizar el desarrollo de la evolución se tiene una pantalla donde se grafican diversos elementos de los individuos seleccionados a lo largo ejecución del algoritmo genético. Estás gráficas pueden mostrar todo el rango de la evolución o limitarse a un intervalo en particular.

4.2.2. *MATLAB*

MATLAB ofrece numerosas herramientas, bibliotecas y un lenguaje de programación orientado al desarrollo de aplicaciones matemáticas, incluyendo el procesamiento de señales. Por esto, se crearon diversas funciones en *MATLAB* que se exponen como una librería que es consumida por la aplicación de *C#*.

4.2.3. Espectrogramas

Una de las principales funciones que se realiza es el cálculo de espectrogramas, que corresponde a la transforma de Fourier de tiempo corto de la señal (STFT), y permite conocer el contenido frecuencial de la señal a lo largo del tiempo. Para este cálculo se utilizó la función nativa de *spectrogram()*, donde los parámetros de entrada incluyen la

señal de audio, la frecuencia de muestreo, y la cantidad de puntos de la transformada (también conocido como tamaño de transformada). Este último parámetro es muy importante ya que determina la cantidad de recipientes o contenedores en los que se divide el espectro de la señal, lo que define la resolución de la misma. Sin embargo, números muy grandes aumentan el costo de procesamiento y el tamaño de los vectores de resultados, lo que a su vez aumenta la complejidad de cálculos posteriores.

Adicionalmente, se tiene como parámetro de entrada una ventana, definiendo el tipo, longitud y porcentaje de traslape. Esta ventana será aplicada a toda la señal de audio para dividirla en tramas. Los efectos de este proceso son bien conocidos en el análisis de señales [Harris (1978)], y existen valores típicos que se utilizan en señales de audio. Por esto, se tiene una ventana por defecto tipo *Hanning*, con una longitud del 2 % de la frecuencia de muestreo, y un traslape del 20 %. No obstante, todos los parámetros del espectrograma pueden ser definidos desde la aplicación de *C#* y ajustados para cada experimento. Los tipos de ventana disponibles son *Hamming*, *Hanning*, y *Blackman-Harris*.

Como parámetros de salida, la función de *spectrogram()* devuelve una matriz con las amplitudes y fase de cada contenedor de frecuencias para cada trama, un vector con la frecuencia central de cada contenedor, un vector con los valores del tiempo y una matriz de potencias para cada contenedor de frecuencias y para cada trama. Como el ser humano no es sensible a cambios de fase [Chu (2003)] y para evitar realizar operaciones con números complejos, en este proyecto se utilizó la matriz de potencias en vez de la amplitud.

Finalmente, se tiene un parámetro de salida que corresponde a la imagen que devuelve el espectrograma. Esta imagen se utiliza para poder actualizar la interfaz de usuario con información de las señales, pero se omite durante el proceso propio de evolución para agilizar los cálculos.

Código 4.1: Función personalizada que calcula el espectrograma.

```
function [ a, b, c, pxx, image ] = GetSpectrogram( xvalues, yvalues, samplingFreq,
                    windowType, windowSize, overlapPercent,
                    numberFFTpoints, plotMatlab )
%   WindowType options:
%       1 = rectangular
%       2 = Hamming
%       3 = Hanning
%       4 = Blackman-Harris
%       -1 = Default window (hanning)
%   Outputs:
%       a = amplitudes
%       b = frequencies
%       c = time vector
%       pxx = power spectral density
%       image = espectrogram image

    image = 1;
    Fs = samplingFreq;

    switch windowType
        case 1
            tmpWindow = windowSize;
        case 2
            tmpWindow = hamming(windowSize);
        case 3
            tmpWindow = hann(windowSize);
        case 4
```

```
            tmpWindow = blackman(windowSize);
        otherwise
    end

    overlap = floor(windowSize * (overlapPercent / 100));

    if (windowType == -1)        %Window default values
        tmpWindow = hann(0.02 * Fs);
        overlap = floor(0.20 * length(tmpWindow));
        numberFFTpoints = 2048;
    end

    [a, b, c, pxx] = spectrogram(yvalues, tmpWindow, overlap, numberFFTpoints,
                Fs, 'yaxis');
end
```

4.2.4. Otras funciones

Los procesos que interactúan directamente con muestras de señales de audio se realizan directamente con funciones de *MATLAB*. Se utilizan funciones nativas para las tareas de lectura, escritura y reproducción de audio, así como para la síntesis de señales y los cálculos de *fitness*. Estos procesos se describen a lo largo de este capítulo.

4.2.5. C#

La solución está dividida en dos proyectos *Audiovolution* y *Core* (ver figura 4.1), donde el primer proyecto contiene las pantallas de la aplicación y el manejo general de errores. El segundo proyecto contiene las clases que representan las señales de audio y clases auxiliares como serilizadores, parámetros y cálculos para la *fitness*. Este proyecto contiene además el punto de acceso a las funciones construidas en *MATLAB*, con la lógica necesaria para invocarlas así como las validaciones y transformaciones de tipos de datos necesarias para hacerlo. Al igual que en el proyecto de *EvoLisa*, la implementación del algoritmo genético se encuentra directamente dentro del código de la pantalla principal.

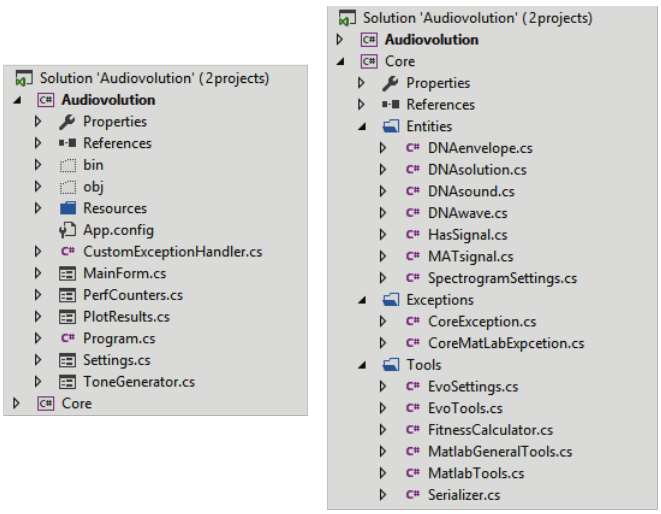

Figura 4.1: Estructura de la solución *Audiovolution*.

4.2.6. Principales Clases

El proyecto *Audiovolution.Core* contiene las clases que representan a los individuos que en este caso son sonidos (ver firgura 4.2). Estos están descompuestos en varias clases donde la clase principal es *DNAsound*. Sus propiedades incluyen la duración de la señal en segundos, la frecuencia de muestreo en hertz y el valor de *fitness* del sonido. Adicionalmente, utiliza una colección de elementos *DNAwave* que corresponden a las primitivas utilizadas para sintetizar la señal, y un elemento *DNAenvelope* que es la envolvente de amplitud general que se aplica a la señal resultante. Estas últimas clases contienen la propiedades manejadas para generar sonidos siguiendo la síntesis aditiva que se explica más adelante en este capítulo.

Tanto *DNAsound* y *DNAwave* heredan de la clase abstracta *HasSignal* la cual contiene una propiedad que es el vector de valores de la señal y un método para generarla. Esto se debe a que para poder realizar las operaciones necesarias en en el programa, tanto los sonidos completos como los parciales (que son las primitivas de audio) deben ser calculados por *MATLAB* antes de poder ser utilizados en otros cálculos. El uso de esta clase abstracta simplifica la construcción del proyecto.

Finalmente, la clase *DNAsolution* contiene una colección de *DNAsound* y es utilizada para guardar la información del proceso de evolución, donde cada elemento de esta colección corresponde a un individuo seleccionado durante la ejecución. Tiene además propiedades que guardar información de los indicadores de evolución, como lo son la cantidad de individuos seleccionados, las generaciones transcurridas, el tiempo transcurrido y una lista que almacena el rendimiento en funciones específicas. Esta clase permite a la aplicación guardar y cargar ejecuciones completas, no sólo el resultado final.

Los métodos que implementan estas clases permiten realizar mutaciones, obtener el vector que representa la señal, e inicializarse. Para *DNAwaves*, la mutación general invoca mutaciones para cada uno de los componentes siguiendo las probabilidades de mutación y el rango de valores determinados por la aplicación. Todas los componentes pueden mutar libremente dentro del rango de valores, excepto la frecuencia la cual está limita a 1 octava de distancia del valor actual (donde 1 octava corresponde al doble de frecuencia). Esto se diseñó así para buscar que la mutación genere individuos cercanos y en los experimentos se encontró que la frecuencia es el principal contribuyente para el cálculo de *fitness*. Para el caso de la inicialización, todos los valores son aleatorios, dentro de los rangos permitidos.

Para *DNAsound*, la mutación invoca las funciones de mutación para cada primitiva *DNAwave* que le pertenece, además de agregar o remover elementos de esta lista. También invoca la mutación de la envolvente *DNAenvelope*, la cual modifica sus valores de manera similar a *DNAwave*. En este caso, el orden de las primitivas no afecta el resultado final de la síntesis, por lo que no hay operaciones para modificarlo.

4.2.7. Interfaz de Usuario

La interfaz de usuario consta de 4 pantallas. La pantalla principal donde se ejecuta y monitorea el experimento, 1 pantalla de configuración de parámetros, 1 pantalla con un generador manual de señales, y 1 pantalla para visualizar los resultados de la evolución.

La pantalla principal (ver figura 4.3) se divide en 3 secciones. Del lado izquierdo, están presentes los elementos que interactúan con la señal objetivo. Aquí es posible

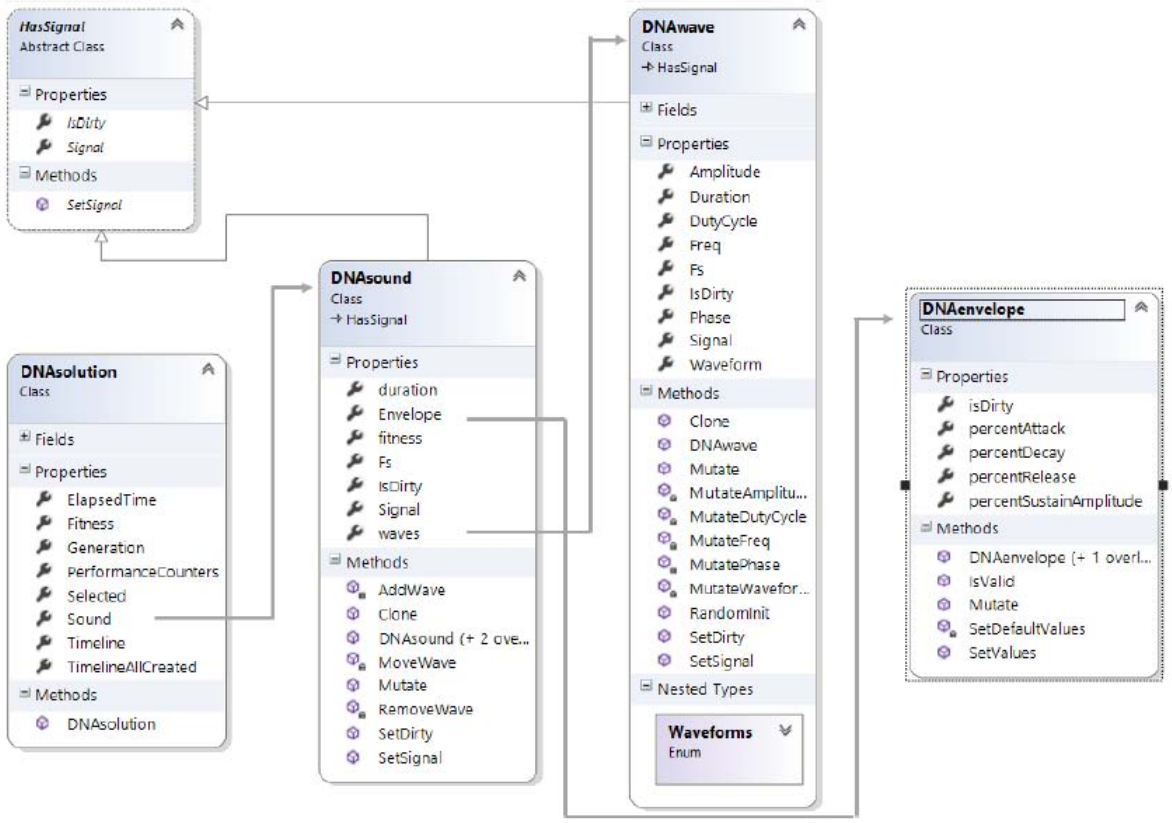

Figura 4.2: Diagrama de clases para *Audiovolution*.

visualizar los datos generales de la señal cargada, los indicadores principales de la ejecución de la evolución, así como botones para iniciar, detener y limpiar la evolución actual. Del lado derecho, se tienen controles y etiquetas para analizar los parámetros de la mejor solución actual. Por último, se tiene una sección en medio donde aparecen gráficas tanto para el sonido objetivo como para el mejor resultado actual. Son dos tipos de gráficas, la primera corresponde al espectrograma tradicional, mientras que la segunda es una gráfica predefinida que muestra el valor de la señal en el tiempo, el cálculo de la DFT sobre toda la señal, y el espectrograma. Esta gráfica ofrece una vistazo rápido sobre los principales puntos de las señales y es útil para conocer elementos básicos de los sonidos, pero no tiene suficiente detalle para sacar conclusiones sobre el experimento. Todos los espectrogramas mostrados utilizan los parámetros definidos en la pantalla de configuración.

La pantalla de configuración (ver figura 4.4) permite asignar todos los parámetros que afectan la ejecución del programa. Se divide en 3 secciones: Tazas de Mutación, Rangos para Sonidos, y espectrograma. Los valores de todos estos parámetros se pueden guardar como un archivo **.settings*, para poder cargarlos posteriormente y usarlos en otras ejecuciones.

La sección de Tazas de Mutación controla los valores que determinan que tan probable es que las diversas operaciones de mutación se lleven acabo. A nivel de *DNAsound*, se manipulan las operaciones de añadir, remover, o mover el orden en la lista para los componentes *DNAwave*. A nivel de *DNAwave*, hay valores para controlar la mutación de la frecuencia, amplitud, fase, ciclo de trabajo y forma de onda. Por último, para las

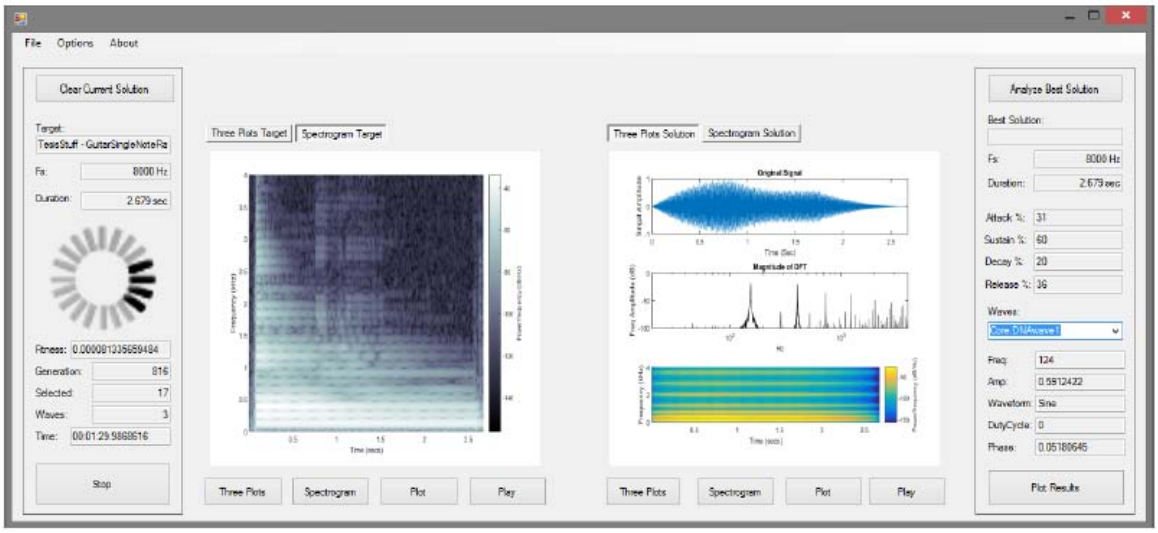

Figura 4.3: Pantalla principal de *Audiovolution*.

envolventes *DNAenvelope*, todas las modificaciones se engloban en un solo valor.

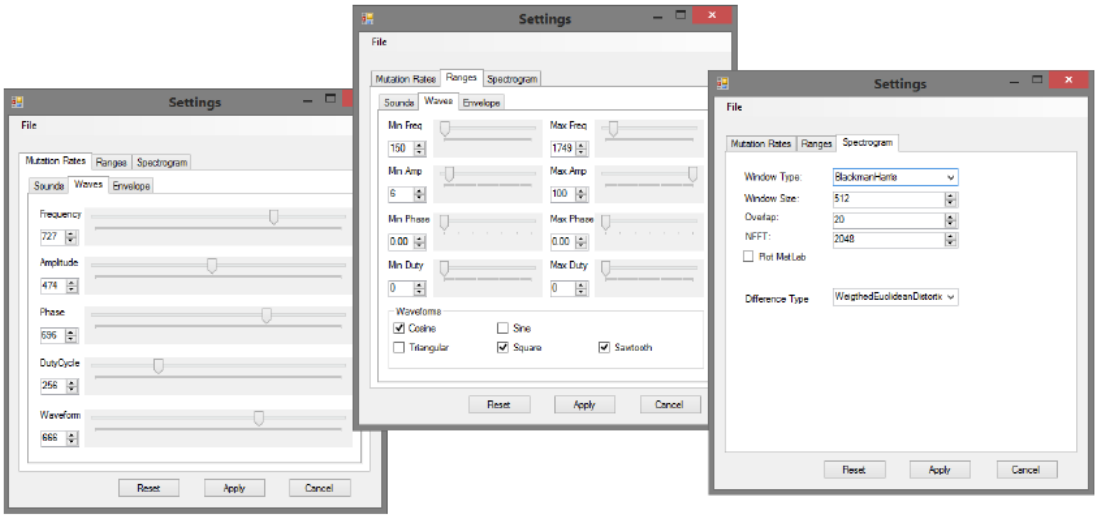

Figura 4.4: Pantalla configuración de *Audiovolution*.

La sección de Rangos para Sonidos ajusta los valores máximos y mínimos que podrán aparecer en los individuos evolucionados, tanto en inicializaciones como en mutaciones. Para los individuos *DNAsound*, se determina la cantidad de *DNAwave* que podrá contener. Para las componentes *DNAwave*, se fija el rango de valores para frecuencia, amplitud, fase, ciclo de trabajo, y es posible seleccionar que formas de onda serán válidas. En este caso, el valor más pequeño que el mínimo de la frecuencia pueda tomar es 20 Hz, debido a que este es la frecuencia más baja que percibe el ser humano (Moore (2007)). El valor más grande que puede tomar el máximo de la frecuencia está determinado por la frecuencia de muestreo del sonido objetivo cargado. Sino se ha cargado un sonido, será 20 000 Hz. Además, para la fase se toman valores en radianes, por lo que un valor de 6,28 corresponde a 2π que equivale a un ciclo completo. Finalmente,

para las envolventes *DNAenvelope*, se controla el rango de valores para los 4 elementos que la componen.

La sección de espectrograma define los parámetros que se usarán en la generación de espectrogramas en toda la aplicación, tanto para las gráficas de visualización como para los cálculos de *fitness* a lo largo de la evolución. Para el tipo de ventana, están disponibles las ventanas más utilizadas en análisis de audio (Hamming, Hanning, Blackman-Harris), además de la rectangular. Adicionalmente, se tiene un parámetro par seleccionar la función de distancia entre espectrogramas que se utilizará para determinar la *fitness* de los individuos. Las funciones posibles se explican más adelante.

La pantalla de Sintetizador Manual (ver figura 4.5) permite generar sonidos con las mismas funciones de síntesis que utiliza la aplicación. Esta funcionalidad es útil para generar tonos de prueba o para crear individuos iniciales a partir de los cuales comenzar una evolución. Contiene todos los parámetros que rigen la síntesis, los cuales se explican en la sección de síntesis. A diferencia del resto de la aplicación, esta pantalla no está limitada por los valores seleccionados en la pantalla de configuración.

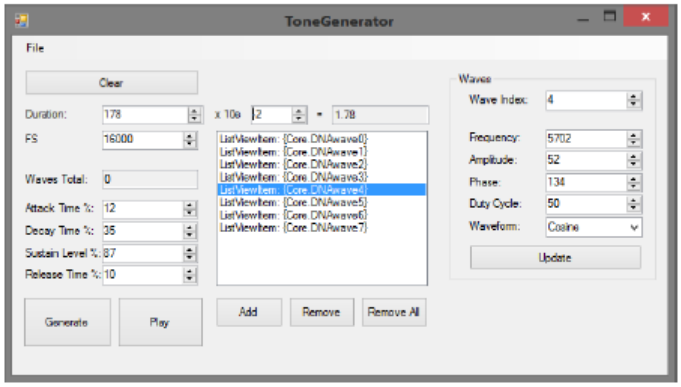

Figura 4.5: Pantalla Sintetizador Manual de *Audiovolution*.

La pantalla de visualización de resultados (ver figura 4.6) tiene herramientas para analizar el proceso de evolución. Permite generar gráficas para observar como fueron cambiando diversos elementos en cada individuo seleccionado (ver figura 4.7). Los datos disponibles para graficar son:

- Señal objetivo o resultante en el tiempo - Ya sea la señal completa o un intervalo de tiempo.
- Evolución de parámetros de la envolvente ADSR
- Evolución de los parámetros de las componentes del *DNA* - Se puede graficar tanto varios parámetros para un solo *DNAwave* de la solución final, un sólo parámetro a lo largo de todos los *DNAwave* de la solución.
- Curva resultante de la envolvente ADSR - De forma similar a como se visualizaría la envolvente en un sintetizador musical.
- Evolución del valor de *fitness* - Para identificar que inviduos producen el mayor cambio en *fitness*.
- Indicadores de rendimiento.

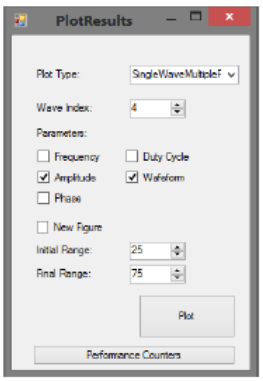

Figura 4.6: Pantalla de visualización de resultados de *Audiovolution*.

La interfaz de usuario en general opera en un hilo diferente a donde se ejecuta el proceso de evolución, por lo que no se bloquea y continua respondiendo mientras opera el algoritmo. Sin embargo, las funciones de *MATLAB* no están optimizadas para trabajar de manera paralela, por lo que sólo es posible hacer un llamado a éstas a la vez, lo que limita la funcionalidad de la interfaz.

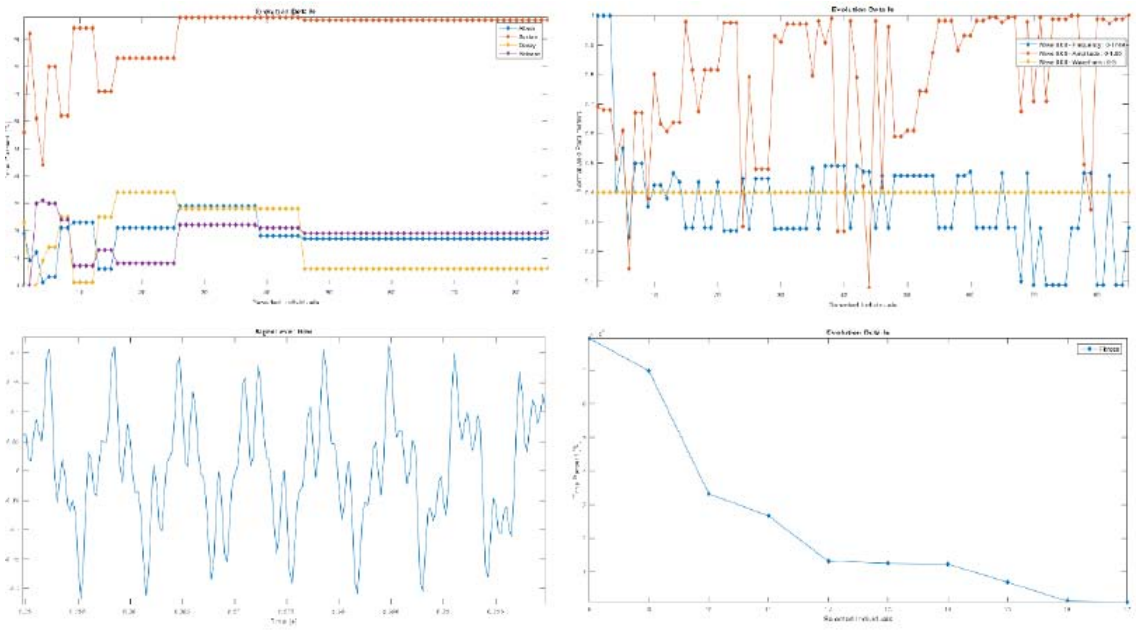

Figura 4.7: Ejemplos de gráficación de resultados

4.2.8. Implementación del Algoritmo Genético

La implementación del algoritmo genético se construyó siguiendo el ejemplo de *EvoLisa* [Johanson (2015)]. El código se ejecuta directamente en la pantalla principal. Primero se verifica que el sonido objetivo haya sido cargado y que los valores para la frecuencia de muestreo y duración sean válidos. En caso de que el mejor resultado actual sea nulo, es decir que sea una evolución nueva, se genera un nuevo sonido con

inicialización aleatoria.

El algoritmo genético se ejecuta dentro de un ciclo *while* limitado por una bandera, de modo que continuará ejecutándose hasta que se detenga manualmente. Dentro del ciclo, se genera un clon del mejor resultado actual, se realiza la mutación del clon, se calcula la *fitness* del clon, y se compara con aquella del mejor resultado actual, por lo que la población total es siempre de 2 individuos. El cálculo de *fitness* es una operación compleja y se describe más adelante cómo se realiza. Al final del ciclo, se asigna como mejor resultado actual aquel individuo que tenga mejor *fitness*. Como en este caso la *fitness* es una medida de error, se seleccionará aquella que sea menor.

Adicionalmente, para conocer el detalle de la evolución, se actualizan dos colecciones del objeto *DNAsolution* durante el ciclo. Por un lado, se agregan todos los individuos generados (aunque sean descartados), y por otro se agregan sólo aquellos que son seleccionados. Estas propiedades son necesarias para la graficación de resultados y son útiles para conocer mayor información sobre algún experimento y no sólo se utilizan para modificar los indicadores de evolución.

Código 4.2: Ciclo del algoritmo genético.

```
private void StartEvolution()
{
    if (target == null)
    {
        string msg = "Error: No target signal has been loaded yet.";
        MessageBox.Show(msg, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }

    fullWatch.Start();
    if (current == null || current.Fs == 0 || current.duration <= 0.0)
    {
        current = new DNAsound(target.Fs, target.duration);
        lastSelected = 0;
        UpdateSolution(current.Clone());
    }

    while (isRunningEvolution)
    {
        DNAsound newSound;
        evolutionCycleComplete = false;
        lock (current)
        {
         newSound = current.Clone();
        }
        newSound.Mutate();
        UpdateSolutionAllCreated(newSound.Clone());

        if (newSound.IsDirty)
        {
         generation++;

         double newErrorLevel = Core.FitnessCalculator.GetSoundFitness(newSound,
                     target, SpectrogramSettings.GetActive());

         if (newErrorLevel < errorLevel)
         {
          selected++;
          lock (current)
          {
              current = newSound;
              UpdateSolution(current.Clone());
          }
```

```
            errorLevel = newErrorLevel;
          }
        }
        evolutionCycleComplete = true;
48    }
}
```

La población se inicializa con un individuo aleatorio, que contiene una sola primitiva *DNAwave*, la cual también se inicializa con parámetros aleatorios. En este caso, no hay ninguna condición que ajuste qué valores pueden tomarse, por lo que el rango de valores posibles sólo está determinado por la pantalla de configuración. A este individuo inicial también se le asigna una envolvente aleatoria.

Para la representación del *DNA* de los individuos, se siguió el enfoque de *EvoLisa*, donde en vez de utilizar un arreglo de bits para capturar todo el genotipo, se distribuyó la información a lo largo de los diferentes clases que componen el individuo. El proceso de mutación se dispara desde el individuo *DNAsound*, el cual ejecuta las mutaciones de las primitivas y envolvente siguiendo las probabilidades de mutación correspondientes, además de que también puede agregar o eliminar primitivas *DNAwae* del individuo. Para el caso de la envolvente *DNAenvelope* todos los parámetros se modificando dentro de la misma operación de mutación, y se verifica para garantizar que todos los valores correspondan a una envolvente válida, donde la suma del ataque, decay y liberación debe ser mayor o igual a 0 y menor o igual a 100 % ($0 <= sum <= 100$). De no cumplirse, se tomará la envolvente por defecto.

4.3. Parámetros de Síntesis

4.3.1. Síntesis Aditiva Modificada

Existe una gran variedad de técnicas y algoritmos para sintetizar sonidos. Para este proyecto se buscó utilizar un tipo de síntesis suficientemente simple para disminuir el costo de procesamiento y que a la vez emulara el concepto visto en *EvoLisa*, donde una señal (en ese caso imágenes) puede ser representada por una colección de primitivas (polígonos con color sólido). Se buscó además que el tipo de síntesis tuviera sustento teórico, pero que al mismo tiempo reflejara tendencias y conceptos musicales. Por todo esto se seleccionó una síntesis aditiva modificada, donde se utiliza el concepto principal de la síntesis aditiva y algunos elementos de la síntesis substractiva.

Desde el punto de vista matemático, de acuerdo al teorema de Fourier, toda señal puede ser representada como una suma de senoidales [Weeks (2011)]. En términos generales, este concepto es aplicado en la síntesis aditiva, donde se generan sonidos complejos combinando senoidales relacionadas, conocidas como señales armónicas [McDermott (2008)]. Asimismo, una de las síntesis más populares utilizadas por sintetizadores musicales es la síntesis substractiva, la cual funciona a partir de primitivas ricas en armónicos generadas por osciladores fijos, los cuales crean señales básicas como senoidales, ondas triangulares o cuadradas, donde después se aplican filtros y modulaciones que manipulan el sonido resultante [McGuire and van der Rest (2016)].

La síntesis de la aplicación funciona a partir de la combinación de primitivas *DNAwave*, que representan ondas simples. Estas ondas a su vez representan la salida de un oscilador en un sintetizador musical. Cada sonido se compone de una colección de estas

ondas simples y una envolvente general. Para cada onda, los parámetros que se pueden modificar son la frecuencia, amplitud, ciclo de trabajo, fase y forma de onda (ver tabla 4.1). Todos los parámetros tiene un rango predeterminado, pero este puede ser acotado desde la pantalla de configuración. Además, no todos los parámetros aplican para todos las formas de onda, donde por ejemplo, el ciclo de trabajo sólo es relevante para ondas cuadradas.

Para sintetizar un sonido, primero se obtiene el vector de valores que representa cada una de las primitivas *DNAwave*. Esto se realiza en *MATLAB* y se utilizan funciones matemáticas, la duración y frecuencia de muestreo para calcular cada una de las muestras. Existen 5 tipos diferentes de formas onda disponibles: senoidal, cosenoidal, triangular, cuadrada (tren de pulsos), y diente de sierra. Estas formas de onda se seleccionaron porque son ampliamente conocidas dentro del mundo musical, donde tienen un carácter asociado, además de que computacionalmente se calculan de manera eficiente.

Código 4.3: Función para generar señales primitivas.

```
function [ n, xvalues, x ] = GetPrimitiveSignal( waveform, duration, frequency,
                          amplitude, samplingFreq, phase, dutyCycle )
%GETPRIMITIVESIGNAL Returns a primitive signal of the specified parameters
%   Generic function to create primitive signals
%   n returns the vector from 0 to samplingFreq * duration, total of samples
%   xvalues returns the values of the x axis, in seconds
%   x returns the actual values of the signal at each sample
%   Waveform values :
%       1 = sine
%       2 = cosine
%       3 = triangular
%       4 = square
%       5 = sawtooth

%Sampling period
Ts = 1/ samplingFreq;
F0 = frequency / samplingFreq;
n = [0 : samplingFreq * duration];
xvalues = [0:Ts:duration];

switch waveform
    case 1 %sine
        x = amplitude * sin( 2 * pi * F0 * n + phase);
    case 2 %cosine
        x = amplitude * cos( 2 * pi * F0 * n + phase);
    case 3 %tringular
        x = amplitude * sawtooth( 2 * pi * F0 * n + phase, 0.5);
    case 4 %square
        x = amplitude * square( 2 * pi * F0 * n + phase, dutyCycle);
    case 5 %sawtooth
        x = amplitude * sawtooth( 2 * pi * F0 * n + phase);
    otherwise
        error('Unknown waveform')
end
```

Una vez que se tienen definidas las muestras de cada primitiva, continua la suma. Ésta es una combinación simple donde se suman los valores de cada muestra para todas las primitivas del sonido. Al final se hace una normalización para garantizar que los valores del resultado estén dentro del rango de -1 a 1, lo que es necesario para que las señales de audio sean procesadas de manera correcta por *MATLAB*. La ecuación de esta combinación es:

$$S_F = \frac{\sum_{i=0}^{L} W_i}{L}$$

donde S_F es la señal combinada, W_i es cada señal primitiva que la compone, y L es la cantidad de señales primitivas. En este caso, el orden en que se combinan no es relevante.

Después de generar la señal combinada, se aplica una envolvente de amplitud que se explica a continuación. Sin embargo, una limitante de este tipo de síntesis es que las primitivas que componen la señal final no tienen variación en el tiempo, dado que la envolvente general se aplica a la señal resultante. Para contra restar esto, muchos sintetizadores musicales utilizan además osciladores dedicados a generar variación en las señales, aplicando filtros o modulaciones en diferentes instantes de tiempo. Para este proyecto no se consideraron este tipo de modificadores, por que no son relevantes para el tipo de señales que se va a procesar, y su implementación aumenta el costo de procesamiento del algoritmo genético ampliamente.

Parámetro	Rango	Unidad
Frecuencia	20 - 20,000	Hz
Amplitud	0 - 100	%
Ciclo de Trabajo	0 - 100	%
Fase	0 - 2π	rad
Forma de onda	sin, cos, tri, saw, sqr	N/A

Tabla 4.1: Parámetros para síntesis aditiva.

4.3.2. Envolvente ADSR

Cuando se produce un sonido, ya sea a partir de un elemento natural o algún instrumento musical, la amplitud del mismo no es constante, sino que tiene variaciones. Por ejemplo, al presionar una tecla de piano el sonido resultante crece hasta alcanzar una amplitud máxima, y después baja en nivel hasta desaparecer. Este tipo de respuesta se conoce como envolvente de amplitud o envolvente de tiempo global. En términos generales, todo sonido puede ser dividido en 3 secciones a lo largo del tiempo. La sección inicial es el ataque, después viene una etapa de estabilidad o estado estable, y finalmente hay una decadencia [Hajda (2007)].

Los límites de estas secciones no son fáciles de definir, ya que no es suficiente identificar el punto donde la amplitud toma el valor máximo. Existen diversos experimentos que proponen mediciones objetivas para identificar estas transiciones. Hajda (2007) propuso el modelo ACT, donde identifica 4 regiones específicas:

- Ataque - Cuando la amplitud RMS de la señal va aumentando.
- Transición Ataque/Estado Estable - Cambio entre el final del ataque y el primer máximo local de amplitud RMS.
- Estado Estable - Segmento donde la amplitud varía alrededor de un valor medio.
- Decadencia - Cuando la amplitud RMS decae rápidamente.

La envolvente de amplitud es también uno de los principales factores que contribuyen a la percepción del timbre de sonidos [Moore (2007), Peeters et al. (2011)]. Es responsable de que algunos sonidos sean secos y percusivos, por ejemplo una marimba o guitarra eléctrica muteada, mientras que otros sonidos son lentos y suaves, como gran parte de los instrumentos de cuerda. Como este proyecto se basa en la similitud del timbre de sñales de audio, es importante considerar esta envolvente y permitir que interactúe con el algoritmo genético de manera ágil.

Una de las implementaciones más utilizadas en sintetizadores musicales es la envolvente ADSR (Attack, Decay, Sustain, Release) [McGuire and van der Rest (2016)]. En esta se controla la longitud de tiempo para los parámetros de Ataque, Decaimiento (decay), y Liberación, mientras que el valor de sustain determina el nivel de amplitud (en decibeles) en el que el sonido se estabilizará. Este tipo de envolvente tiene la ventaja de que interactúa muy bien con controladores musicales como por ejemplo teclados. El ataque y decaimiento marcan el tiempo que el sonido generado tarda en llegar al nivel de sustain donde se estabilizará a partir del momento en que se presiona una tecla, mientras que la liberación actúa después de que se deja de presionar la tecla. Adicionalmente, la variación en la amplitud no sigue un comportamiento lineal. Para el caso de la envolvente ADSR, típicamente se usan curvas parabólicas para el ataque, decaimiento y liberación.

Para este proyecto, se implementó una envolvente ADSR que se aplica a la señal resultante (*DNAsound*), por lo que ésta modula la combinación final de señales primitivas. Los parámetros de esta envolvente se muestran en la tabla 4.2, y a lo largo de esta tesis se representan como un vector de 4 cantidades, que siguen la forma [ataque, decay, sustain, liberación]. Un ejemplo de la envolvente generada se muestra en la figura 4.8.

Parámetro	Rango	Unidad
Ataque	0 - 100	% tiempo
Decay	0 - 100	% tiempo
Sustain	0 - 100	% amplitud
Liberación	0 - 100	% tiempo

Tabla 4.2: Parámetros de la envolvente general de amplitud.

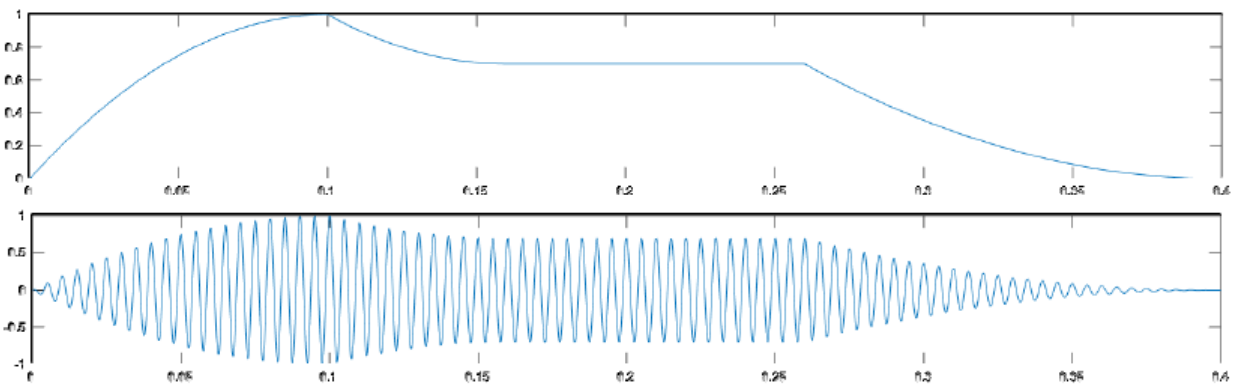

Figura 4.8: La imagen superior muestra la curva total de la envolvente, mientras que la imagen inferior muestra la envolvente aplicada a una senoidal.

4.4. Definición de función de *fitness*

Una de las decisiones más críticas para el desarrollo de un sistema basado en algoritmos genéticos es la función de *fitness*, ya que esta función será la encargada de seleccionar qué individuos o soluciones son mejores que otros, y con eso determinar el éxito de la evolución. Debido a la naturaleza iterativa del cómputo genético, se debe considerar además el rendimiento y el costo computacional de esta función. Hay ocasiones donde una función de *fitness* menos precisa pero menos costosa da mejores resultados ya que es posible ejecutar más ciclos. Además, estas funciones dependen del campo en el que se aplican por lo que no existe un estándar.

Para este proyecto, se consideraron una variedad de diferentes operaciones. Se tomaron en cuenta dos formas diferentes para plantear estas funciones. En primer punto se buscaron métodos para obtener un vector que caracteriza cada señal de audio. En segundo punto se buscaron funciones de distancia para medir la diferencia entre estos vectores. La comparación de señales de audio es una tarea compleja, debido a que el oído humano no tiene una percepción lineal del sonido, y existen varios fenómenos psicoacústicos que influyen en la percepción. Por esto, se consultó la literatura para seleccionar aquellas funciones de *fitness* que han dado buenos resultados y se realizó un pequeño estudio de éstas para así determinar la más adecuada para la aplicación. Las funciones consideradas se describen a continuación.

Pointwise - La diferencia entre cada muestra de la señal objetivo y el candidato [McDermott et al. (2008)] . Esta medida normalmente no es muy útil para comparar señales de audio, ya que se basa en una comparación de las señales en el tiempo e ignora el contenido de frecuencia. Además se ha demostrado experimentalmente [McDermott et al. (2005)] que el uso de esta función en algoritmos genéticos hace que la evolución de individuos tienda al silencio, debido a la influencia de la fase. Se consideró en el experimento sólo como una referencia. Esta función está definida por la ecuación:

$$d_p(x, y) = \frac{\sum_{t=0}^{T} |x_t - y_t|}{2T}$$

donde x, y son las señales de audio, y T es la longitud de las mismas. En caso de que no sean de la misma longitud se complementa con 0's la señal más corta.

DFT - La DFT corresponde a la transformada discreta de Fourier de la señal completa, y en este caso se hacen 3 cálculos con diferente cantidad de puntos y se obtiene la sumatoria [McDermott et al. (2005)]. Esta medida ya refleja el dominio de la frecuencia, sin embargo, no considera la variación que las señales puedan tener en el tiempo. Es decir, sólo aplica cuando la señal es estable en el tiempo. Se consideró en el experimento sólo como una referencia. Esta función está definida por la ecuación:

$$d_F(x, y) = \frac{\sum_{L \in \{256,1024,4096\}} d_{F_L}(x, y)}{3}$$

donde cada transformada de Fourier está definida por:

$$d_{F_L}(x, y) = \frac{\sum_{j=0}^{N} \left(\sum_{i=0}^{L/2} |X_j(i) - Y_j(i)| \right)}{N}$$

donde x, y son las señales de audio, y N es la cantidad de transformadas (en este caso es 3). $X_j(i)$ y $Y_j(i)$ corresponden a la magnitud normalizada de la transformada de Fourier j para la banda i. El tamaño de la transformada está fijo en 256, 1024, o 4096 y sólo se cosnideró la parte positiva.

Características - Existen una variedad de atributos que se pueden extraer a partir de señales de audio y que sirven para describirlas. Individualmente estas características proveen información demasiado especializada como para poder realizar comparaciones significativas, pero al considerar diversas características en conjunto se obtiene una representación útil. No obstante, no todos estos atributos son apropiados para todos los campos de estudio, por lo que es necesario identificar y considerar sólo aquellos que si lo son. Para esta tesis se seleccionó sólo una pequeña muestra de características que son importantes para la percepción del timbre de sonidos, utilizadas en experimentos previos por McDermott et al. (2008) entre otros, y que además son relevantes al tipo de señales que se emplean. Algunas de estas características se calculan a partir de la longitud completa de la señal, mientras que otras se hacen a partir de la señal dividida en tramas, o incluso a partir de las tramas del espectrograma. Las características empleadas son:

- **Amplitud promedio al cuadrado** - Se calcula la amplitud promedio de toda la señal. Determinado por McDermott et al. (2005) como:

$$amp(x) = \frac{\sum_{t=0}^{T} \left|x(t)\right|}{T}$$

  donde T es la longitud de la señal en muestras.

- **Envolvente** - Como se mencionó anteriormente, la envolvente de señales musicales tiene gran importancia en la percepción de las mismas. Para calcularla se utiliza el algoritmo de Hilbert que da una buena aproximación [The Mathworks Inc. (2016), Hartmann (2007) y Peeters et al. (2011)]. Este cálculo se hace sobre la señal completa y devuelve un vector de valores.

- **Centroide Espetral** - Es una medida simple que refleja el centro de gravedad del espectro de la señal [Peeters et al. (2011)], lo que indica en que área del espectro se concentra mayor energía. Siguiendo la implementación de Giannakopoulos and Pikrakis (2014), esta medida se define por:

$$C = \frac{\sum_{k=1}^{L/2} kX_i(k)}{\sum_{k=1}^{L/2} X_i(k)}$$

  donde $X_i(k)$ es la amplitud de la transformada de Fourier para el contenedor k de la i-ésima trama, y L es el número de puntos de la misma (y por lo tanto de contenedores).

- **Propagación Espectral** - Representa que tanto se aleja el espectro de la señal del centroide espectral y mide como se distribuye la energía. Funciona como la desviación estándar del espectro [Peeters et al. (2011)]. Se calcula siguiendo con la ecuación propuesta por Giannakopoulos and Pikrakis (2014) de la siguiente manera:

$$S = \sqrt{\frac{\sum_{k=1}^{F}(k - C_i)^2 X_i(k)}{\sum_{k=1}^{F} X_i(k)}}$$

donde F es la longitud de la trama en muestras, $X(k)$ es la amplitud de la transformada de Fourier para el contenedor k de la i-ésima trama y C_i es el centroide espectral para la trama i.

- **Entropía Espectral** - Mide la forma en que se distribuye el espectro, y refleja que tantos "picos."están presentes [Misra et al. (2004)]. Está definida por la ecuación:

$$H = -\sum_{F=0}^{L-1} n_f \cdot log_2(n_f)$$

donde n_f es la energía total del espectro, y L es la cantidad de bandas (contenedores) del espectrograma.

- **MFCCs** - Una de las formas más populares para caracterizar señales de voz es mediante el uso de los Coeficientes Cepstral de Frecuencia-Mel (MFCC por sus siglas en inglés). Estos coeficientes son usados para representar la envolvente espectral de señales de audio [Rocha et al. (2013)]. Para obtener estos coeficientes, se aplica una serie de filtros a la señal, se calcula la potencia de la señal dentro de cada filtro, y después se aplica la transformada cosenoidal discreta [Müller et al. (2011)]. El resultado es un vector de valores en la escala Frecuencia-Mel, la cual considera la percepción humana del sonido. En esta aplicación, se utilizó la implementación del algoritmo dado por Giannakopoulos and Pikrakis (2014) para obtener los coeficientes de cada trama, y posteriormente se utilizaron la media y desviación estándar de cada coeficiente para generar el vector que caracteriza a la señal. Cómo es típico, se utilizaron sólo los primeros 13 coeficientes, aunque este puno no es definitivo y no existe consenso sobre cuantos coeficientes es necesario usar.

Después de obtener todas las características, se genera un vector el cual caracteriza a la señal completa. Este vector contiene los valores de las características y para aquellos que se obtienen a partir de tramas, se obtiene la media y al desviación estándar. La composición final de este vector se describe en la tabla 4.3. Finalmente, la distancia entre diferentes vectores de características se realiza mediante la ecuación:

$$d_{carac}(A, B) = \sum_{i=1}^{35} \frac{A(i) - B(i)}{A(i)}$$

donde $A(i)$ y $B(i)$ son la i-ésima característica para las señal A y B respectivamente. Esta distancia considera la diferencia de cada valor como un porcentaje del valor original, lo que evita que valores muy grandes (como suelen presentarse en los coeficientes bajos del MFCC) dominen fuertemente.

Diferencia al cuadrado - A partir del espectrograma de ambas señales, se calcula la diferencia al cuadrado. Este método es simple e eficiente pero no toma en cuenta

Índice	Característica	Tipo
1	Centroide Espectral	mean
2	Centroide Espectral	std
3	Propagación Espectral	mean
4	Propagación Espectral	std
5	Entropía Espectral	mean
6	Entropía Espectral	std
7	Envolvente Amplitud	mean
8	Envolvente Amplitud	std
9	Amplitud media	N/A
10	MFCC(1)	mean
11	MFCC(1)	std
12	MFCC(2)	mean
13	MFCC(2)	std
14	MFCC(3)	mean
15	MFCC(3)	std
16	MFCC(4)	mean
17	MFCC(4)	std
18	MFCC(5)	mean
19	MFCC(5)	std
20	MFCC(6)	mean
21	MFCC(6)	std
22	MFCC(7)	mean
23	MFCC(7)	std
24	MFCC(8)	mean
25	MFCC(8)	std
26	MFCC(9)	mean
27	MFCC(9)	std
28	MFCC(10)	mean
29	MFCC(10)	std
30	MFCC(11)	mean
31	MFCC(11)	std
32	MFCC(12)	mean
33	MFCC(12)	std
34	MFCC(13)	mean
35	MFCC(13)	std

Tabla 4.3: Vector de características de una señal.

ninguna particularidad de la percepción humana del audio y penaliza los errores grandes más gravemente que los pequeños [Loizou (2013)]. Está definida por:

$$d_d(X_k, Y_k) = (X_k - Y_k)^2$$

donde X_k y Y_k representan el valor del espectrograma para cada señal de audio, en específico corresponde a la matriz de potencias.

Medida de distorsión Euclideana Ponderada - A partir del espectrograma de ambas señales, se calcula la diferencia dividiendo entre el valor del espectrograma original. Esto tiene la ventaja de que el resultado de la diferencia representa un porcentaje y no un valor aislado, por lo que da el mismo peso tanto a frecuencias que tienen mucha amplitud como aquellas que tienen poca [Loizou (2013)]. Está definida por:

$$d_{WEDM}(X_k, Y_k) = \frac{(X_k - Y_k)^2}{X_k}$$

donde X_k y Y_k representan el valor del espectrograma para cada señal de audio, en específico corresponde a la matriz de potencias.

Distancia Itakura-Saito - La distancia de Itakura-Saito ha sido utilizada exitosamente en numerosas aplicaciones de procesamiento de voz para realizar comparaciones. En esta tesis se utilizó la implementación de Loizou (2013), dada por la ecuación:

$$d_{is_k}(X_k, Y_k) = \frac{X_k^2}{Y_k^2} - \log\left(\frac{X_k^2}{Y_k^2}\right) - 1$$

donde X_k y Y_k representa el valor del espectrograma de la matriz de potencias para el recipiente de frecuencias k.

Medida de Cosh - Una de las desventajas de la medida de *Itakura-Saito* es que no siempre es simétrica, es decir, que $d_{is}(X_k, Y_k) \neq d_{is}(Y_k, X_k)$. Para resolver esto, se creó la medida de Cosh, que combina dos formas de la distancia de Itakura-Saito [Loizou (2013)]. Esta distancia está definida por:

$$d_{cosh}(X_k, Y_k) = cosh\left(\log\frac{X_k}{Y_k}\right) - 1 = \frac{1}{2}\left[\frac{X_k}{Y_k} + \frac{Y_k}{X_k}\right] - 1$$

donde X_k y Y_k representa el valor del espectrograma de la matriz de potencias para el recipiente de frecuencias k.

Proporción de probabilidad ponderada - Esta medida da mayor peso a los errores cerca de las frecuencias con magnitud alta. La implemantación propuesta por Loizou (2013) está definida por:

$$d_{WLR}(X_k, Y_k) = (\log X_k - \log Y_k)(X_k - Y_k)$$

donde X_k y Y_k representa el valor del espectrograma de la matriz de potencias para el recipiente de frecuencias k.

Distancia Espectral Logarítmica - Otra medida simple para calcular la distancia entre espectrogramas [Goldberg and Reik (2000)], que toma en cuenta la percepción logarítmica de frecuencias pero no realiza ningún ajuste y da el mismo peso a todo el rango de frecuencias. Está definida por la ecuación:

$$d_{log}(X_k, Y_k) = \sum \left| \log|X_k|^2 - \log|Y_k|^2 \right|$$

donde X_k y Y_k representa el valor del espectrograma de la matriz de potencias para el recipiente de frecuencias k.

4.5. Experimentos

4.5.1. Evaluación de *fitness*

Para poder determinar cuál función de *fitness* es la más adecuada se realizó un experimento comparando las funciones mencionadas anteriormente. El objetivo de este experimento no es obtener un estudio profundo sobre la relación entre las diversas funciones y la percepción subjetiva de señales de audio, ya que esto es un tema complejo y material de estudios más avanzados.

Este experimento se dividió en 3 etapas, donde cada etapa corresponde a una señal de audio. En cada etapa, se generó una señal de audio objetivo (S_O) directamente en *MATLAB* (a partir de funciones matemáticas nativas) con parámetros determinados. A continuación, se generaron 4 señales ($S_{B0}, S_{B1}, S_{B2}, S_{B3}$) con el sintetizador manual de la aplicación *Audiovolution*, donde S_{B0} contiene los mismos parámetros que S_O , mientras que S_{B1}, S_{B2} y S_{B3} contienen elementos de S_O y si van alejando progresivamente de S_O. Es decir, que S_{B1} es una señal muy cercana (desde el punto de vista de parámetros) a S_O ; S_{B2} es una señal poco cercana a S_O y S_{B3} es una señal lejana a S_O.

Adicionalmente, se generaron dos señales a modo de control. S_A utiliza los mismos parámetros que S_O, pero fue generada a partir de las funciones de síntesis de audio directamente desde *MATLAB*. S_{noise} corresponde a un ruido blanco con distribución Gausiana. Para S_A, el error resultante debe ser pequeño y representa el mejor caso posible, mientras que S_{noise} debe tener un error grande. Del mismo modo, para ($S_{B0}, S_{B1}, S_{B2}, S_{B3}$) se espera que el error sea pequeño para S_{B0} y se vaya incrementándose con S_{B1}, S_{B2}, y S_{B3}.

Para la ejecución del experimento, se aplicó una segmentación por ventana tipo *Hanning* de 512 muestras de largo, con un traslape de 102 muestras (lo que corresponde al 20 % del tamaño de la ventana). No todos las funciones que se calcularon utilizan esta segmentación, ya que hay algunas que operan directamente sobre la señal completa, como es el caso de DFT y PointWise.

La primera señal objetivo utilizada en este experimento, consiste de una sola senoidal con frecuencia $F_0 = 900$ Hz. Esta señal no tiene envolvente ni cambios en el tiempo. Las señales de prueba modifican la frecuencia acercándose a F_0, de modo que S_{B3} está a 2 octavas de distancia, S_{B2} está a 1 octava, S_{B1} a está 1/2 octava y S_{B0} está justo a F_0, manteniendo los demás parámetros con los mismos valores que S_O (ver la tabla 5.1).

La segunda señal consiste de 8 senoidales con diferentes frecuencias y amplitudes, sin envolvente ni cambios en el tiempo. En este caso, las señales de prueba modifican la cantidad de senoidales, manteniendo los parámetros individuales de cada senoidal. Los parámetros iniciales se definieron de manera aleatoria (ver la tabla 4.5).

Para la tercera señal objetivo, se utilizó la misma señal con 8 senoidales, pero se aplicó una envolvente ADSR por lo que se introduce cambio en el tiempo. Las señales de prueba contienen las 8 senoidales con los mismos valores que en S_O, pero van modificando los parámetros de la envolvente aproximándose a los valores reales (ver la tabla 5.3).

La representación en el tiempo para la primer señal de prueba se muestra en las figuras 4.9 y 4.10. Aparece tanto la señal original como todas las señales cercanas para poder identificar la transición entre cada paso. Para la segunda señal se muestran en las figuras 4.11 y 4.12, mientras que las de la tercera señal están en las figuras 4.13 y 4.14.

4.5.2. Calibración de *Audiovolution*

Después de observar las propiedades de las funciones de *fitness* en el experimento anterior, se realizó un experimento para probar el funcionamiento de la aplicación de *Audiovolution* y poder evaluar el desempeño de la evolución. Además de conocer la influencia de los parámetros configurables (rangos y probabilidades de mutación). Para esto, se generaron 8 señales de prueba (S_{C0} hasta S_{C7}) y por cada señal, se ejecutó el

Señal	Origen	Frencuencias	Amplitudes	Envolvente ADSR
S_O	MATLAB	900	100	[0 0 100 0]
S_A	Síntesis MATLAB	900	100	[0 0 100 0]
S_{B0}	Sintesis C#	900	100	[0 0 100 0]
S_{B1}	Sintesis C#	675	100	[0 0 100 0]
S_{B2}	Sintesis C#	1800	100	[0 0 100 0]
S_{B3}	Sintesis C#	3600	100	[0 0 100 0]
S_{noise}	MATLAB	N/A	100	[0 0 100 0]

Tabla 4.4: Señales para el primer experimento de fitness.

Señal	Origen	Frencuencias	Amplitudes	Envolvente ADSR
S_O	MATLAB	0715 1891 2322 4137	86 62 52 60	[0 0 100 0]
		5702 6131 6524 7012	52 85 14 61	
S_A	Síntesis MATLAB	0715 1891 2322 4137	86 62 52 60	[0 0 100 0]
		5702 6131 6524 7012	52 85 14 61	
S_{B0}	Sintesis C#	0715 1891 2322 4137	86 62 52 60	[0 0 100 0]
		5702 6131 6524 7012	52 85 14 61	
S_{B1}	Sintesis C#	0000 1891 0000 4137	00 62 00 60	[0 0 100 0]
		0000 6131 0000 7012	00 85 00 61	
S_{B2}	Sintesis C#	0000 0000 0000 4137	00 00 00 60	[0 0 100 0]
		0000 6131 0000 0000	00 85 00 00	
S_{B3}	Sintesis C#	0000 0000 0000 4137	00 00 00 60	[0 0 100 0]
		0000 0000 0000 0000	00 00 00 00	
S_{noise}	MATLAB	N/A	100	[0 0 100 0]

Tabla 4.5: Señales para el segundo experimento de fitness.

Señal	Origen	Frecuencias	Amplitudes	Envolvente ADSR
S_O	MATLAB	0715 1891 2322 4137	86 62 52 60	[14 25 78 32]
		5702 6131 6524 7012	52 85 14 61	
S_A	Síntesis MATLAB	0715 1891 2322 4137	86 62 52 60	[14 25 78 32]
		5702 6131 6524 7012	52 85 14 61	
S_{B0}	Síntesis C#	0715 1891 2322 4137	86 62 52 60	[14 25 78 32]
		5702 6131 6524 7012	52 85 14 61	
S_{B1}	Síntesis C#	0715 1891 2322 4137	86 62 52 60	[15 26 75 32]
		5702 6131 6524 7012]	52 85 14 61	
S_{B2}	Síntesis C#	0715 1891 2322 4137	86 62 52 60	[10 20 100 30]
		5702 6131 6524 7012	52 85 14 61	
S_{B3}	Síntesis C#	0715 1891 2322 4137	86 62 52 60	[0 0 100 0]
		5702 6131 6524 7012	52 85 14 61	
S_{noise}	MATLAB	N/A	100	[0 0 100 0]

Tabla 4.6: Señales para el tercer experimento de fitness.

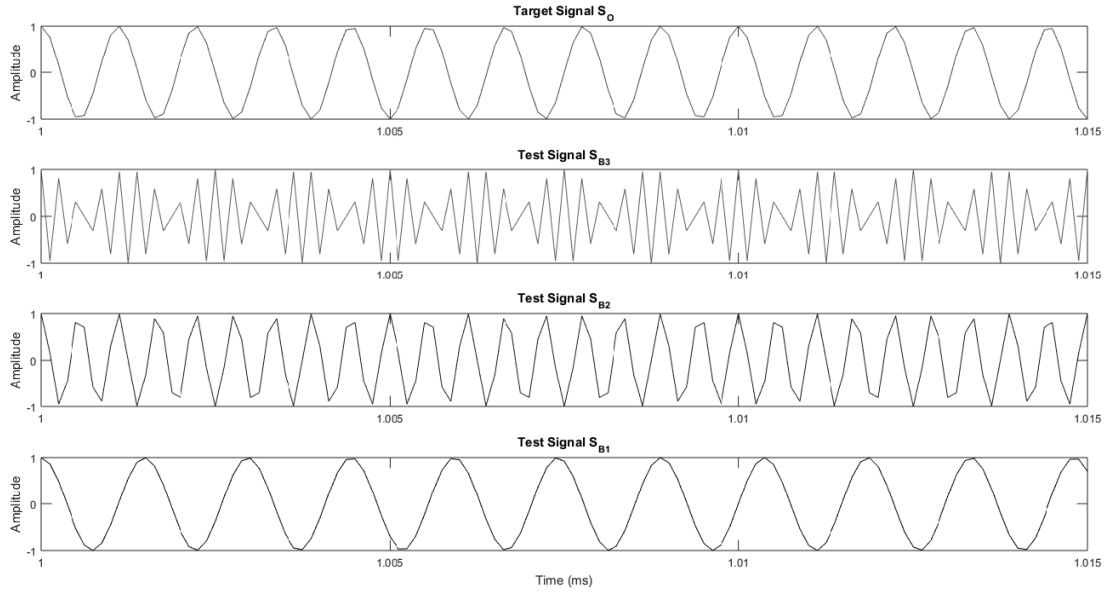

Figura 4.9: Evaluación de *fitness* - Señales en el tiempo para la primera señal de prueba.

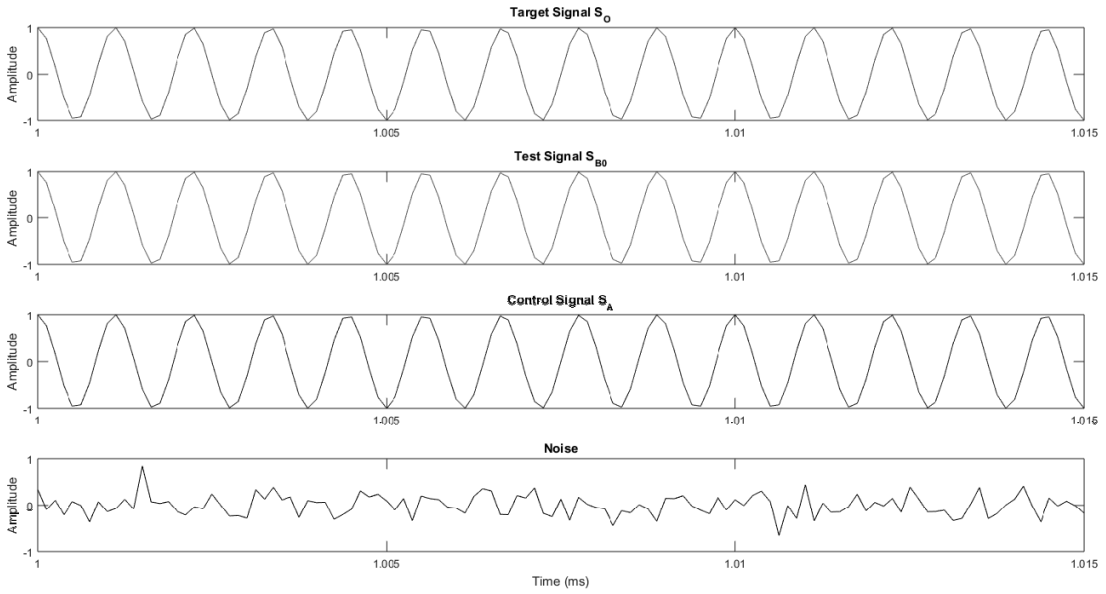

Figura 4.10: Evaluación de *fitness* - Señales en el tiempo para la primera señal de prueba.

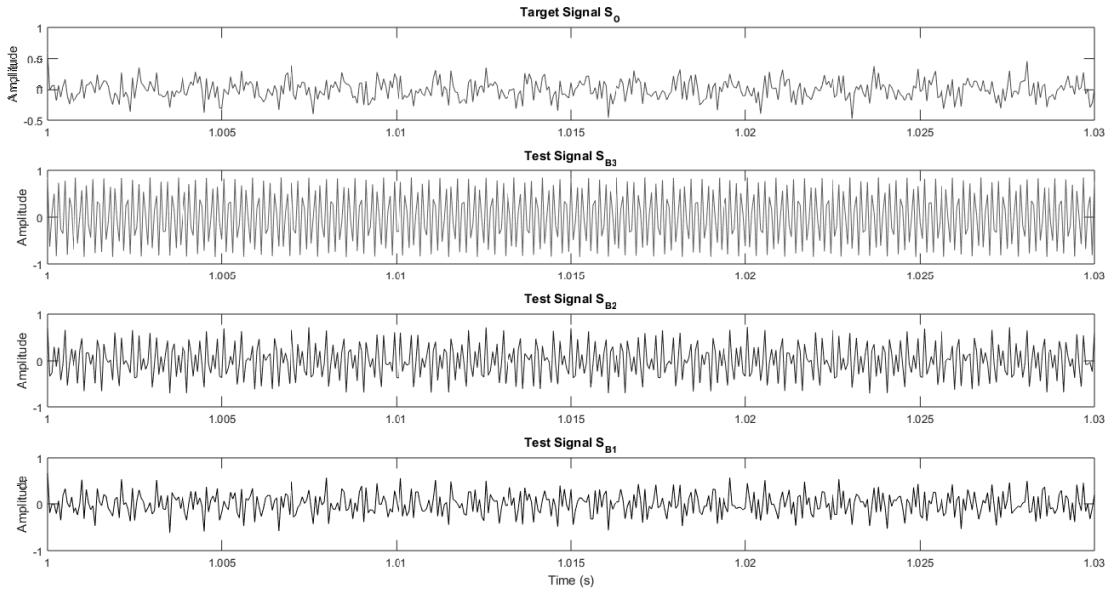

Figura 4.11: Evaluación de *fitness* - Señales en el tiempo para la segunda señal de prueba.

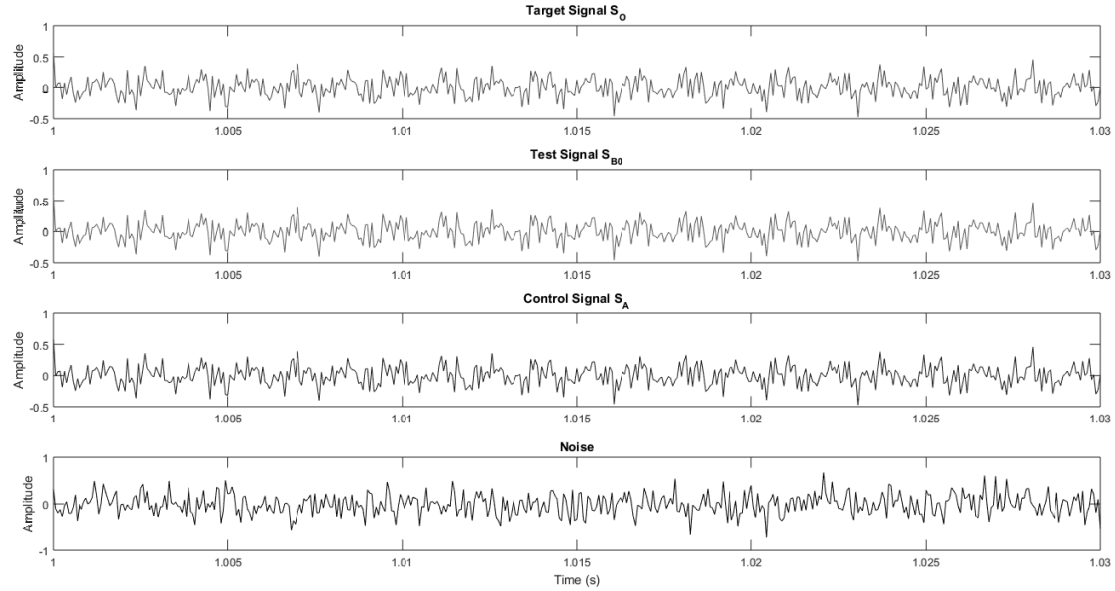

Figura 4.12: Evaluación de *fitness* - Señales en el tiempo para la segunda señal de prueba.

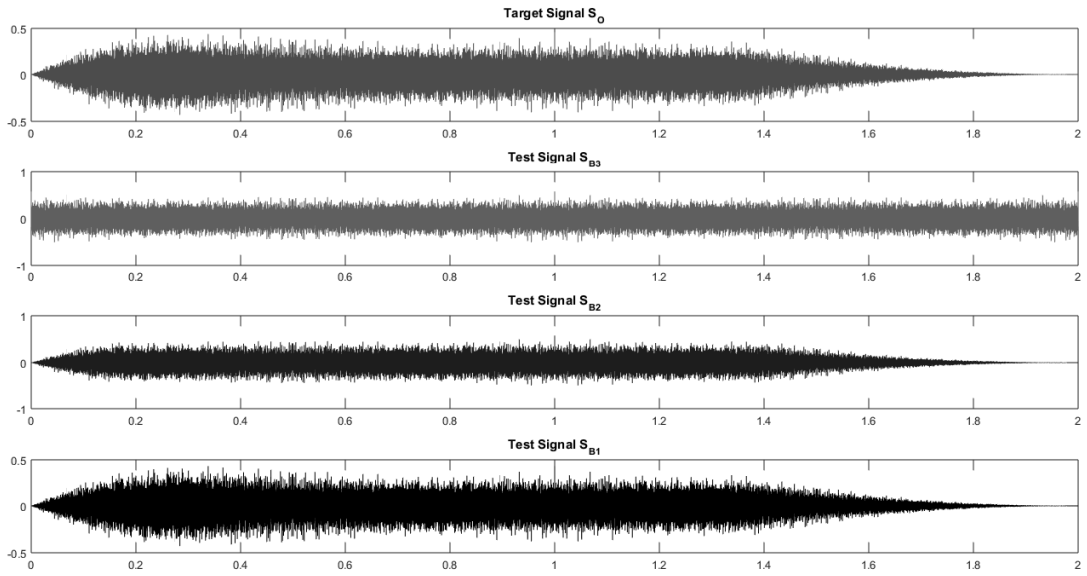

Figura 4.13: Evaluación de *fitness* - Señales en el tiempo para la tercera señal de prueba.

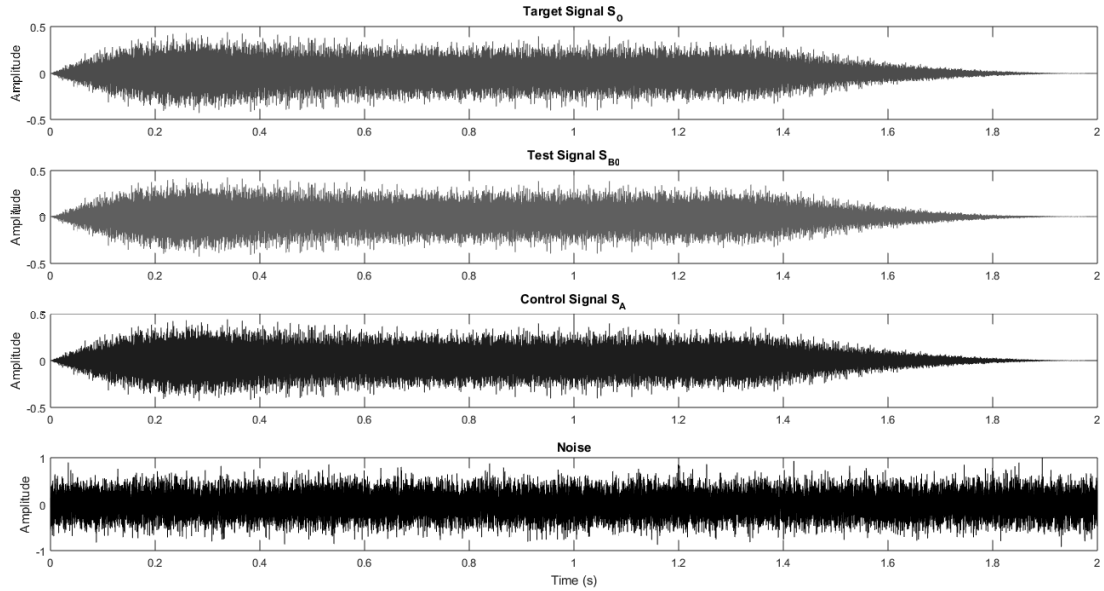

Figura 4.14: Evaluación de *fitness* - Señales en el tiempo para la tercera señal de prueba.

algoritmo genético del programa 5 veces con el fin de replicar el sonido. Los parámetros de la evolución se ajustaron al tipo de señal, y se fueron modificando entre ejecuciones buscando obtener resultados consistentes. La evolución se dejó operar hasta que el sonido resultado fuera subjetivamente idéntico al original o en su defecto hasta que la *fitness* calculada se estabilizara, es decir que se seleccionaran nuevos individuos en 1 minuto.

Las señales de prueba utilizadas fueron generadas directamente en *MATLAB* utilizando las funciones de síntesis creadas. En general, estas señales son sencillas y van aumentando en complejidad. Entre cada señal se buscó aislar un sólo componente, donde por ejemplo S_{C3} y S_{C4} son idénticas excepto por la envolvente. Los parámetros utilizados para estas señales se muestran en la tabla 4.7.

Señal	Freq	Amp	Fase	Forma	Envolvente
S_{C0}	320	100	0	1	[0 0 100 0]
S_{C1}	320	100	0	1	[13 26 84 31]
S_{C2}	679	100	0	4	[5 15 78 34]
S_{C3}	679	100	0	5	[5 15 78 34]
S_{C4}	155 234 410	1 56 76	0 0 0	1 1 1	[13 26 84 31]
S_{C5}	155 234 410	1 56 76	1 2 3	1 1 1	[13 26 84 31]
S_{C6}	155 234 410	1 56 76	1 2 3	1 3 4	[13 26 84 31]
S_{C7}	125 398 578 824 1523	100 60 80 75 29	1 2 3 4 5	1 2 3 4 5	[18 15 83 25]

Tabla 4.7: Señales de prueba para la calibración de *Audiovolution*.

Debido a que la frecuencia de muestreo se relaciona directamente con el tamaño de los vectores generados y por tanto en la cantidad de cálculos necesarios, la frecuencia de muestreo de estas señales es pequeña, teniendo 1000, 2000 y 4000 Hz como máximo.

4.5.3. Replicación de sonidos - Sintetizador Musical Simple

A continuación, se ejecutó un experimento utilizando sonidos generados con un sintetizador musical. Esto con el objetivo de evaluar la replicación de sonidos más complejos y que provienen de fuentes musicales y no de funciones matemáticas. Se generaron 3 sonidos de prueba con diferentes parámetros y por cada sonido se ejecutó el algoritmo genético 5 veces. Los parámetros de la evolución se fijaron de acuerdo al tipo de señal y a los resultados del experimento anterior, sin embargo, se ejecutaron unas evoluciones de prueba para asegurar que el conjunto de parámetros se desempeñara de forma correcta y que la evolución si llegara a un resultado aceptable.

Las señales de prueba fueron generadas con el sintetizador virtual *FabFilter Twin 2*, el cual utiliza 3 osciladores, filtros y modulaciones para producir sonidos [FabFilter Software Instruments (2016)]. Este instrumento tiene una gran cantidad de parámetros y opciones que no fueron usados para este experimento, donde se limitó a 1 solo oscilador y 1 envolvente de amplitud. Los valores para ejecutar el sintetizador se fijaron directamente desde midi, para evitar variaciones propias de la ejecución humana. La frecuencia de muestreo usada fue 8000 Hz, lo cual es suficientemente alto para conservar el rango completo del espectro de los sonidos empleados.

La primera señal S_{D0} corresponde a la nota C3 (con una frecuencia fundamental de 261 Hz aproximadente) durante 1 segundo, usando 1 oscilador con forma de onda senoidal y sin envolvente, es decir que el sustain es el 100 % y los demás parámetros son 0. Esta señal es la mas simple que puede generar este sintetizador, pero al ser un instrumento virtual se espera encontrar artefactos y algunas diferencias con respecto a una senoidal pura generada a partir de una función matemática. La segunda señal S_{D1} tiene los mismos parámetros que S_{D0}, pero cambia la envolvente de amplitud. Para S_{D3} se cambio la forma de onda y se usó diente de sierra.

Señal	Frecuencia	Forma	Envolvente
S_{D0}	261	senoidal	[0 0 100 0]
S_{D1}	261	senoidal	[10 15 100 20]
S_{D3}	261	diente de sierra	[10 15 100 20]

Tabla 4.8: Señales de prueba para la replicación de sintetizador musical simple.

4.5.4. Replicación de sonidos - Sonidos Complejos

Por último, se ejecutó un experimento que busca replicar sonidos complejos, generados con instrumentos musicales físicos (no virtuales). La complejidad de sonidos utilizados en este experimento va mucho más allá de experimentos anteriores, ya que estos sonidos incluyen una rica colección de armónicas y formantes, además de imperfecciones y contaminación de fuentes externas como lo son la habilidad del ejecutante, la calidad de los instrumentos, y técnicas de grabación. La síntesis utilizada es limitada y no está preparada para sintetizar correctamente señales de este tipo, por lo que el objetivo de este experimento es conocer el alcance de la aplicación construida.

Se utilizaron 3 señales de prueba (ver tabla 4.9) y por cada señal se ejecutó el algoritmo genético 5 veces. En este caso, tanto los parámetros de configuración como el tiempo de ejecución fueron más abiertos. Además, la inicilización de la población no fue aleatoria, sino que se generó manualmente un sonido inicial utilizando el sintetizador manual. Para este sonido inicial, se analizó la DFT de la señal de prueba y se replicaron las frecuencia fundamental y las 4 primeras armónicas, usando ondas senoidales.

Señal	Fuente	FM	Descripción
S_{Z0}	Piano	11 000	Nota singular con intensidad media y duración de 2 segundos.
S_{Z1}	Guitarra Eléctrica	8 000	Nota singular con intensidad media y duración de 2 segundos.
S_{Z2}	Guitarra Eléctrica.	8 000	Multiples notas.

Tabla 4.9: Señales de prueba para la replicación de sonidos complejos.

La primera señal S_{Z0} corresponde a una nota de piano tocada con intensidad media (*mezzoforte*) durante aproximadamente 2 segundos. Esta señal fue generada a partir del instrumento virtual *Addictive Keys*, el cual utiliza una vasta biblioteca de grabaciones

de pianos físicos para habilitar un piano virtual XLN Audio AB (2016). La segunda señal S_{Z1} es una única nota de guitarra eléctrica tocada con intensidad media y modulación de frecuencia (*vibrato*) ligera. Esta fue capturada a partir de la salida directa del instrumento y después se procesó mediante un simulador de amplificadores, lo que es una técnica común para grabaciones de guitarra. Por último, la señal S_{Z2} también corresponde a una grabación de guitarra eléctrica generada con la misma técnica, pero el sonido es polifónico y consta de varias notas diferentes al mismo tiempo.

Todas estas señales fueron grabadas con una frecuencia de muestreo de 48 000 Hz y una profundidad de bits de 24 bits/s, lo que sigue el estándar para estudios de grabación. Posteriormente se analizó el espectro de cada señal para identificar el rango que contiene información relevante, y se ajustó la frecuencia de muestreo de cada una para reducirla. Las grabaciones fueron editadas digitalmente para eliminar el silencio antes y después de los eventos, debido a que no contribuyen al carácter de los sonidos generados y que la síntesis utilizada no contempla periodos de silencio.

Capítulo 5

Análisis de Resultados

5.1. Evaluación de *fitness*

Uno de los primeros resultados que se obtuvieron fue que al comparar los espectrogramas entre señales sintetizadas desde la aplicación y aquellas sintetizadas con los mismos parámetros pero directamente desde *MATLAB*, existían diferencias pequeñas. Al inspeccionar los valores de las muestras se encontró que estas diferencias se deben al cambio de tipo de datos. Para poder ejecutar las funciones de síntesis desde la aplicación de C# es necesario cambiar el tipo de datos (operación conocida como *casteo*) entre los tipos de datos propios de *MATLAB* y aquellos de *.net*. Además, en *C#* los tipos de datos *float* y *double* no son 100 % precisos, sobre todo al representar números muy pequeños [Griffiths (2012)].

En los espectrogramas, estas diferencias se presentan principalmente como ruido de bajo nivel, donde por ejemplo, para una señal sintetizada directamente en *MATLAB*, la amplitud del espectrograma para una frecuencia F_0 es 0, mientras que para la misma señal pero sintetizada desde la aplicación tendrá un valor pequeño, pero diferente a 0. Se encontró además que estas diferencias son mínimas, donde la mayoría de estos valores están por debajo de los -100 dB usando la escala logarítmica. En la percepción de audio, sonidos a niveles tan bajos no son completamente identificables (por ejemplo, para calcular los tiempos de reverberación se utiliza la regla de RT60, donde se mide el tiempo que un sonido tarda en bajar 60 dB). Por lo que se implementó una funcionalidad para que al momento de calcular las distancias entre espectrogramas, se ignore todo lo que está debajo de un punto predeterminado (*threshold*), el cuál se fijó a -120 dB. Para estos experimentos se consideraron tanto las distancias que usan este threshold como aquellas que no.

Todos los resultados de estos experimentos miden la distancia entre dos señales, usando la función de distancia de *fitness* indicada. Por lo tanto las cantidades deben mostrar valores pequeños para señales que se parecen y valores grandes para señales que no se parecen, considerando que lo importante no son los valores absolutos, sino la relación entre estos. Para el conjunto de señales S_{B3} hasta S_{B0}, se espera ver una clara transición donde el valor de la distancia va disminuyendo, mientras que los valores para S_A y S_{noise} son sólo de control y se espera que S_A muestre el valor más pequeño de todos (y que S_{B0} se aproxime a éste) y que S_{noise} sea un valor grande aunque no necesariamente sea el mayor valor de todos. También se muestra el tiempo que tomó en procesar cada función, incluyendo la operación para obtener el vector de la señal (ya

sea el espectrograma o características) y el cálculo de la distancia.

5.1.1. Experimento 1

En la tabla 5.1 se muestran los resultados de la primera señal. Tanto para la función Pointwise como la DFT, resalta que la diferencia entre S_{B3} y S_{B2} se incrementa, cuando se esperaba que disminuyera, por lo que se comprueba que no son medidas confiables. La función a base de características presenta un caso similar entre S_{B2} y S_{B1} lo cual sugiere que las características empleadas no son una buena representación de este tipo de señales.

Para la función de diferencia al cuadrado, no hay diferencia cuando utiliza el threshold de -130 dB o no, excepto por el tiempo de ejecución. Esto es de esperarse ya que el uso del threshold sustituye los valores pequeños por el valor mínimo que se aproxima a 0, y son tan pequeños que no son significativos para el resultado final. Lo que resalta es que no hay diferencia entre S_{B3}, S_{B2} y S_{B1}. Esto se debe a que al utilizar una sola senoidal, las frecuencias de éstas no se acercan lo suficiente a la frecuencia de S_O como para considerarse cercana.

La distancia WEDM presenta buenos resultados. Cuando no se usa threshold, hay un cambio significativo entre S_{B3}, S_{B2} y S_{B1} seguido de un cambio grande para llegar a S_{B0}. A pesar de que esta última es muy diferente al valor de S_A la trayectoria general es buena. Cuando se utiliza el threshold, tiene un comportamiento similar, excepto que S_{B3} y S_{B2} no tienen cambio, lo que reduce la efectividad de la distancia para señales con poco parecido al objetivo.

Tanto las distancias de Itakura-Saito como la de COSH siguen un comportamiento similar a la WEDM, donde el uso de threshold presenta las mismas deficiencias. La medida de COSH además no tiene una diferencia tan grande entre S_{B1} y S_{B0}, lo que podría dar problemas al momento de discernir los detalles finos entre señales muy parecidas.

La distancia WLR sin threshold, muestra un diferencia muy pequeña entre S_{B3}, S_{B2} y S_{B1}, mientras que la la transición a S_{B0} es muy grande. Cuando se usa threshold no hay cambio significativo entre las señales. Finalmente, para la medida LOG sucede el mismo caso que para la diferencia al cuadrado, donde las tres primeras son prácticamente iguales seguido de un cambio súbito en S_{B0}.

En cuanto a los tiempos de ejecución, el uso de threshold tiene un gran costo de rendimiento, donde en general aumenta el tiempo entre 3 o 12 veces con respecto a la misma distancia sin threshold. El cálculo de las características fue la operación más costosa mientras que Pointwise fue la menor. Esto es esperado ya que incluyen la mayor y menor cantidad de cálculos respectivamente. Del resto, la diferencia al cuadrado, WEDM y COSH tienen un rendimiento similar, al igual que Itakura-Saito, WLR y COSH.

Las distancias que mejor se comportaron para esta señal de prueba fueron la WEDM, Itakura-Saito, y COSH, todas sin utilizar threshold. De estas, la más eficiente fue WEDM, y la más costosa fue Itakura-Saito.

Las figuras 4.9 y 4.10 del capítulo anterior muestran una sección de las señales en el tiempo y los espectrogramas se aprecian en las figuras 5.1 y 5.2. El espectrograma para S_O muestra claramente la ubicación de la única senoidal, y se mantiene a lo largo de toda la duración. Existe un poco de esparcimiento a frecuencias contiguas, pero los niveles decaen rápidamente. Para S_{B3} aparee la componente en la frecuencia esperada,

sin embargo, también se identifican componentes en bajas frecuencias a distancias casi constantes. Estas son armónicos que fueron reflejadas debido a la baja frecuencia de muestreo, y fueron producidos por errores de redondeo al utilizar las funciones de síntesis desde la aplicación de C#. Para S_{B2} estas armónicas inesperadas aparecen en frecuencias superiores e inferiores a la fundamental. Par S_{B1} y S_{B0} están presentes pero en niveles menores y las distancias no son constantes. Todas estas armónicas inesperadas en señales S_B tienen una amplitud muy baja, donde apenas superan los -100 dB. Por esto no afectan fuertemente la percepción de las señales, pero explican las diferencias encontradas para los valores de *fitness* entre S_{B0} y S_A.

Function	S_{B3}	S_{B2}	S_{B1}	S_{B0}	S_A	S_{noise}	Tiempo
Pointwise	0.404,4	0.413,61	0.406,75	$7.439{,}1\times10^{-06}$	$7.840{,}1\times10^{-14}$	0.328,02	1.077,3
DFT	0.380,1	0.453,33	0.393,84	1.065×10^{-05}	1.601×10^{-13}	0.802,92	3.235,4
Características	5.715,5	0.565,19	3.712,8	0.431,29	0.000,131,04	$2.841{,}2\times10^{+11}$	58.904
Dif. Cuadrardo (Thld.)	0.149,47	0.149,47	0.149,47	$3.240{,}8\times10^{-12}$	$6.462{,}5\times10^{-29}$	0.074,609	17.391
Dif. Cuadrardo	0.149,47	0.149,47	0.149,47	$3.240{,}8\times10^{-12}$	$6.462{,}5\times10^{-29}$	0.074,609	1.235,2
WEDM (Thld.)	$1.633{,}3\times10^{+22}$	$1.633{,}3\times10^{+22}$	$5.112{,}8\times10^{+09}$	5,803.3	$4.613{,}9\times10^{-25}$	$3.059{,}9\times10^{+18}$	17.75
WEDM	$9.759{,}2\times10^{+15}$	$2.352\times10^{+13}$	$1.940{,}6\times10^{+09}$	0.001,216,3	$2.511{,}8\times10^{-24}$	$3.933{,}2\times10^{+13}$	1.433,8
Itakura-Saito (Thld.)	$3.569{,}5\times10^{+45}$	$3.569{,}5\times10^{+45}$	$1.701{,}8\times10^{+44}$	$5.780{,}7\times10^{+23}$	$1.507{,}7\times10^{-13}$	$2.331{,}3\times10^{+10}$	28.15
Itakura-Saito	$5.044{,}5\times10^{+32}$	$1.067{,}7\times10^{+27}$	$1.847{,}3\times10^{+22}$	$7.851\times10^{+05}$	$3.317{,}5\times10^{-06}$	$2.331{,}2\times10^{+10}$	11.818
COSH (Thld.)	$1.063\times10^{+24}$	$1.063\times10^{+24}$	$3.271{,}9\times10^{+22}$	$2.519{,}8\times10^{+14}$	$2.309{,}3\times10^{-14}$	$4.963{,}5\times10^{+22}$	19.67
COSH	$4.613{,}3\times10^{+17}$	$1.003{,}7\times10^{+15}$	$6.607{,}7\times10^{+11}$	$7.48\times10^{+08}$	$8.295{,}7\times10^{-07}$	$6.717\times10^{+17}$	2.466,4
WLR (Thld.)	480.86	480.87	194.76	6.569×10^{-08}	$4.613{,}9\times10^{-25}$	56.375	31.233
WLR	330.11	266.6	187.09	$2.141{,}9\times10^{-08}$	$2.511{,}8\times10^{-24}$	47.754	10.823
LOG (Thld.)	$4.246{,}4\times10^{+07}$	$4.189{,}7\times10^{+07}$	$1.321{,}4\times10^{+07}$	$8.356{,}2\times10^{+05}$	0.000,514,18	$8.521{,}5\times10^{+07}$	27.502
LOG	$2.451{,}2\times10^{+07}$	$1.551{,}2\times10^{+07}$	$1.031{,}1\times10^{+07}$	$4.364{,}8\times10^{+06}$	2.046,7	$3.576\times10^{+07}$	12.639

Table 5.1: Resultados para la primera señal de prueba. - Se muestran las diferencias de *fitness* entre la señal S_O y la señal indicada en cada columna, además del tiempo de ejecución. Las cantidades son sólo para comparación y no representan ninguna unidad física.

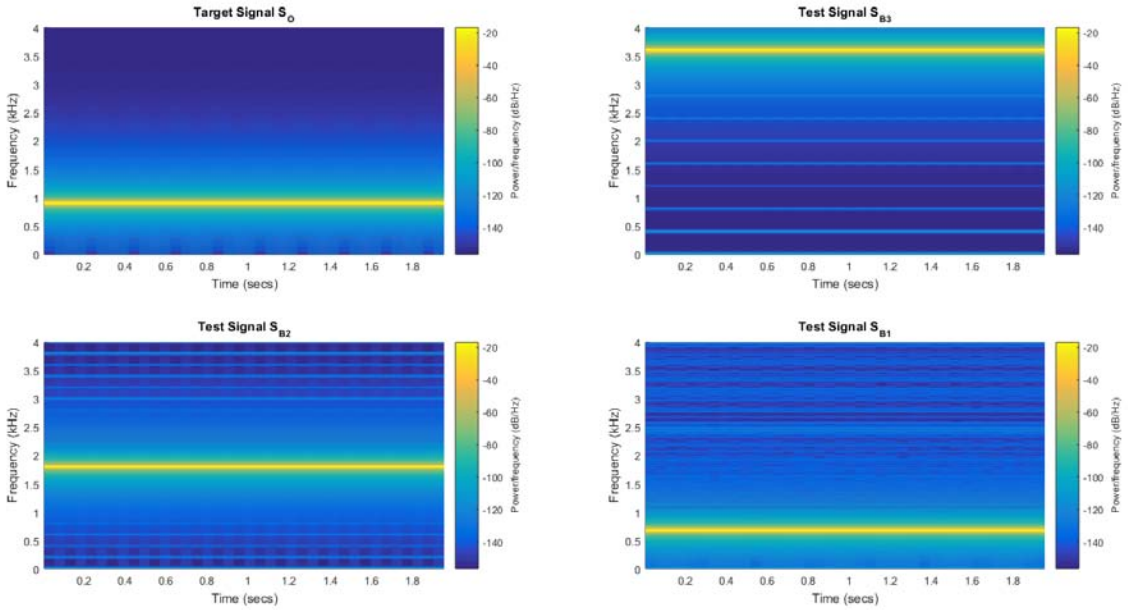

Figura 5.1: Evaluación de *fitness* - Espectrogramas para primera señal de prueba.

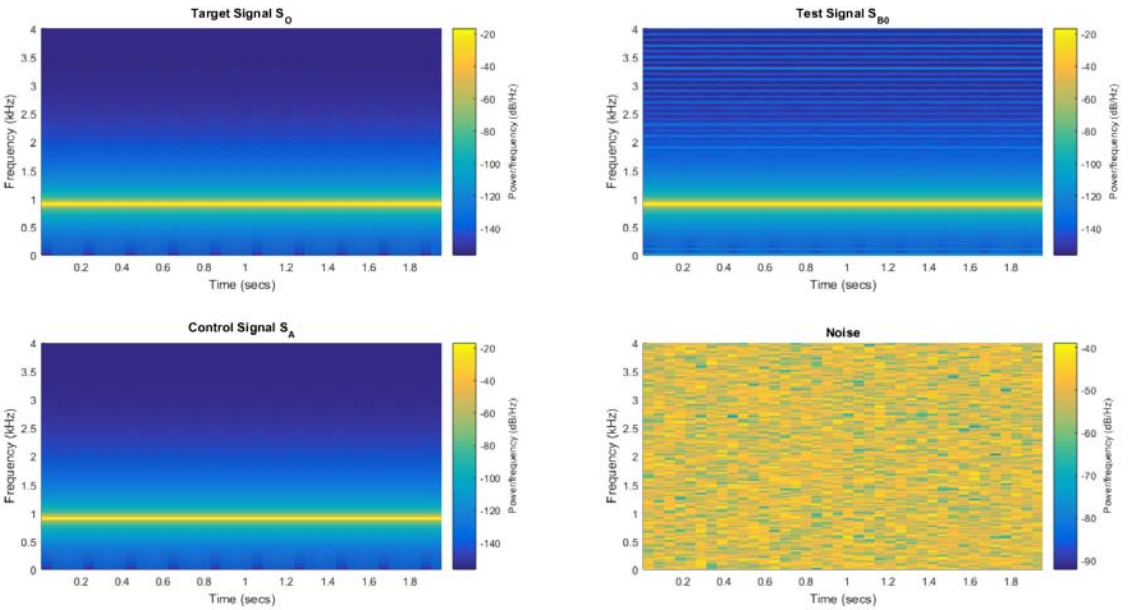

Figura 5.2: Evaluación de *fitness* - Espectrogramas para primera señal de prueba.

5.1.2. Experimento 2

Los resultados para la segunda señal aparecen ene la tabla 5.2 mientras que las señales se aprecian en las figuras 4.11 y 4.12. En este caso, tanto la distancia Pointwise como la DFT presentan el comportamiento buscado, donde la distancia disminuye conforme se acerca a S_{B0}, sin embargo el cambio entre las 3 primeras distancias es pequeño. Para la función de características, se presentó el resultado contrario al buscado, donde la distancia va aumentando en lugar de disminuir. Esto soporta la idea del experimento anterior de que el vector de característica seleccionado no es una representación fiel de este tipo de señales de audio.

La diferencia al cuadrado una vez más no presenta diferencia entre la versión que usa threshold y la que no, sin embargo, en este caso la distancia disminuye como se espera para las señales S_B. Como esta señal de prueba está compuesta de varias senoidales y donde varía la cantidad de las mismas presentes, esta distancia calcula correctamente la similitud a la señal original. Es importante destacar que ninguna de las señales generadas contiene elementos ajenos a la señal objetivo, es decir, que no tienen senoidales con frecuencias diferentes. Por lo tanto, no hay información "basuraz todo el espectro contribuye a la semejanza.

Tanto la distancia WEDM como Itakura-Saito operaron de manera similar al experimento anterior, siguiendo los cambios esperados cuando no se usa threshold. Por otro lado, la distancia COSH sin threshold en esta ocasión presenta una diferencia muy pequeña entre las 3 primeras señales, lo que reduce su efectividad. Finalmente, las distancias WLR y LOG presentan una variación mínima en las primeras señales seguido de un cambio muy grande al llegar a S_{B0}, comportamiento contrario al buscado para la función de *fitness*.

El rendimiento sigue la tendencia marcada por la primera señal de prueba, donde la distancia Pointwise y por características tienen el valor menor y mayor respectivamente. La diferencia al cuadrado, WEDM y COSH tienen un valor similar que tiende a ser bajo, mientras que Itakura-Saito, WLR y LOG presentan un valor cercano pero más grande. Es importante no comparar estos valores de tiempo directamente entre experimentos o señales de prueba, ya que estas cantidades no representan nada físicamente (por ejemplo milisegundos), y los valores dependen de las condiciones locales del equipo de prueba.

Las distancias que mejor se comportaron para la segunda señal de prueba fueron la diferencia al cuadrado, WEDM, e Itakura-Saito, todas sin utilizar threshold. De estas, la más eficiente fue la diferencia al cuadrado seguida de cerca por WEDM, mientras que la más costosa fue Itakura-Saito.

Los espectrogramas visibles en las figuras 5.3 y 5.4 muestra que para la señal objetivo S_O, además de tener componentes para todas las senoidales que componen la señal, existe un ligero movimiento de energía entre 3 y 4kHz. Para el conjunto de señales S_B, se aprecia lo esperado donde aparecen las componentes de las senoidales que componen cada una. Además existen artefactos de bajo nivel a lo largo de todo el espectro. Una vez más estos artefactos son producto de los errores de redondeo al utilizar la síntesis, y su nivel es suficientemente bajo para no afectar la percepción de las señales. Para S_{B0}, se nota también que no está presente el ligero movimiento de energía entre 3 y 4kHz detectado en S_O.

Function	S_{B3}	S_{B2}	S_{B1}	S_{B0}	S_A	S_{noise}	Tiempo
Pointwise	0.240,83	0.124,71	0.063,006	$7.616{,}8\times10^{-06}$	2.198×10^{-13}	0.113,35	1.501,7
DFT	0.132,49	0.080,124	0.044,074	$1.288{,}3\times10^{-05}$	$6.886{,}8\times10^{-14}$	0.349,25	3.336,1
Características	0.410,99	0.713,24	1.915,3	2.891,8	$1.026{,}2\times10^{-08}$	47,952	115.95
Dif. Cuadrardo (Thld.)	0.019,16	0.001,364,2	$8.442{,}7\times10^{-05}$	$4.707{,}2\times10^{-15}$	$8.829{,}9\times10^{-30}$	$1.928{,}2\times10^{-05}$	30.814
Dif. Cuadrardo	0.019,16	0.001,364,2	$8.442{,}7\times10^{-05}$	$4.707{,}2\times10^{-15}$	$8.829{,}9\times10^{-30}$	$1.928{,}2\times10^{-05}$	1.603,5
WEDM (Thld.)	$3.911\times10^{+06}$	$2.865{,}6\times10^{+05}$	45,149	3,387	$1.855{,}2\times10^{-24}$	$6.786{,}5\times10^{+17}$	30.239
WEDM	220.99	18.913	1.374,5	$1.942{,}3\times10^{-06}$	$3.582{,}3\times10^{-24}$	$9.069{,}4\times10^{+08}$	2.073,7
Itakura-Saito (Thld.)	$4.390{,}3\times10^{+41}$	$4.001{,}6\times10^{+41}$	$2.582\times10^{+41}$	$1.035{,}4\times10^{+25}$	2.82×10^{-13}	$1.341{,}9\times10^{+10}$	54.303
Itakura-Saito	$3.737{,}2\times10^{+28}$	$4.014{,}9\times10^{+26}$	$2.565{,}2\times10^{+25}$	$4.567{,}8\times10^{+06}$	$8.978{,}9\times10^{-10}$	$1.341{,}8\times10^{+10}$	22.959
COSH (Thld.)	$1.689{,}8\times10^{+22}$	$1.490{,}4\times10^{+22}$	$7.448{,}3\times10^{+21}$	$3.070{,}8\times10^{+14}$	$3.663{,}7\times10^{-14}$	$2.500{,}6\times10^{+22}$	36.13
COSH	$1.721{,}9\times10^{+14}$	$5.199{,}9\times10^{+13}$	$1.655{,}9\times10^{+13}$	$2.976{,}3\times10^{+05}$	$2.241{,}1\times10^{-10}$	$8.205{,}9\times10^{+13}$	4.612
WLR (Thld.)	21.802	9.766,6	4.101,6	$7.839{,}8\times10^{-08}$	$1.855{,}2\times10^{-24}$	13.283	48.876
WLR	18.491	6.855,5	2.622,2	$1.231{,}3\times10^{-08}$	$3.582{,}3\times10^{-24}$	7.989,5	21.161
LOG (Thld.)	$8.064{,}6\times10^{+07}$	$7.725{,}2\times10^{+07}$	$4.684{,}3\times10^{+07}$	$2.018{,}6\times10^{+06}$	0.001,822,9	$1.059{,}2\times10^{+08}$	51.587
LOG	$2.496{,}7\times10^{+07}$	$2.267{,}1\times10^{+07}$	$1.370{,}4\times10^{+07}$	$1.509{,}2\times10^{+06}$	0.077,674	$4.072{,}1\times10^{+07}$	22.569

Table 5.2: Resultados para la segunda señal de prueba. - Se muestran las diferencias de *fitness* entre la señal S_O y la señal indicada en cada columna, además del tiempo de ejecución. Las cantidades son sólo para comparación y no representan ninguna unidad física.

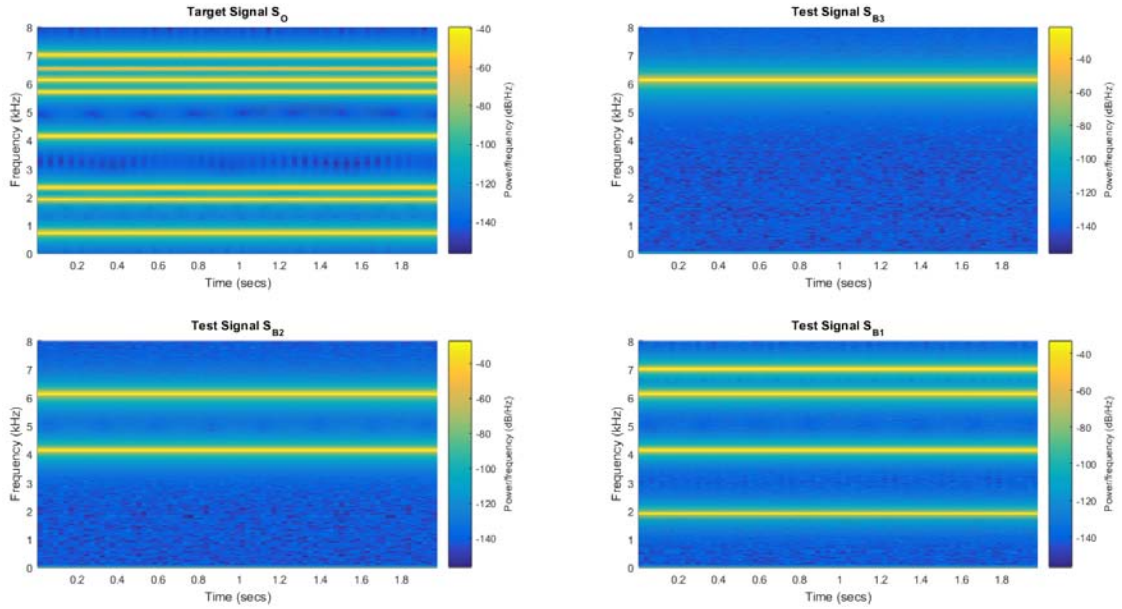

Figura 5.3: Evaluación de *fitness* - Espectrogramas para segunda señal de prueba.

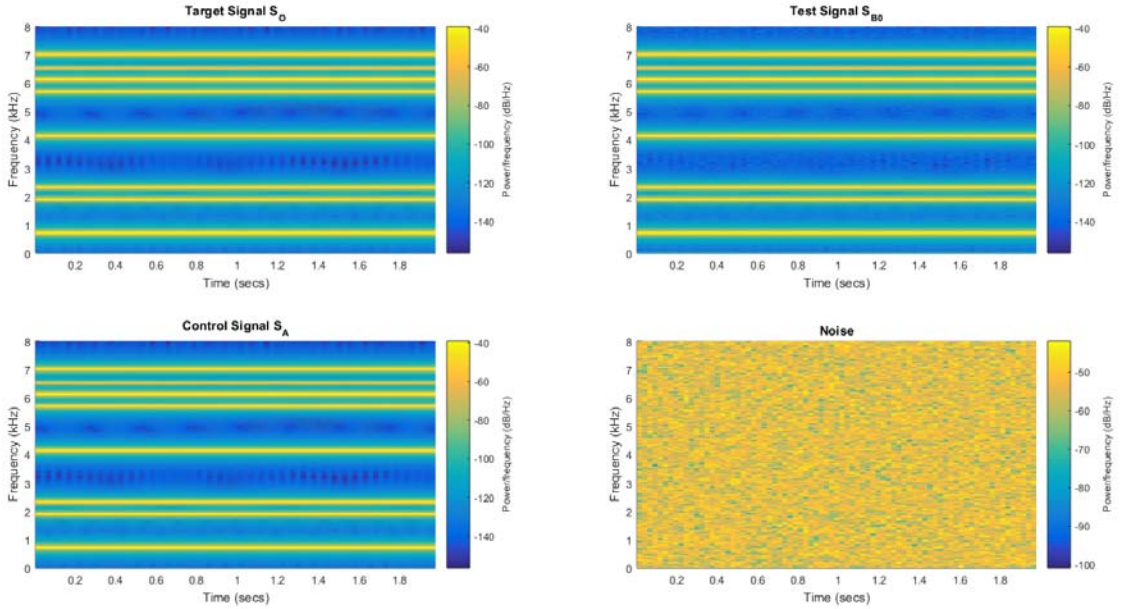

Figura 5.4: Evaluación de *fitness* - Espectrogramas para segunda señal de prueba.

5.1.3. Experimento 3

Para la tercera señal los resultados se muestran en la tabla 5.3 y las figuras 4.13, 4.14 muestran las señales en el tiempo. Tanto la distancia Pointwise y DFT siguen la trayectoria buscada, aunque la diferencia entre las señales S_B es pequeña. Una vez más la distancia por características presenta un comportamiento contrario al esperado, donde el valor obtenido va aumentando. Esto confirma lo visto en experimentos anteriores, esta función no es correcta para este tipo de señales.

La diferencia al cuadrado y la distancia WEDM se desempeñan adecuadamente, sin embargo, la diferencia entre S_{B1} y S_{B0} es muy pequeña, lo que sugiere que el ajuste fino de los parámetros de la envolvente no afectan tanto al resultado final, siempre y cuando esté dentro de valores cercanos. Además, la distancia al cuadrado tampoco distingue marcadamente entre S_{B3} y S_{B2}, esto implica que para tener un aumento considerable de *fitness* es necesario que todos los valores de la envolvente se acerquen al objetivo.

Para el caso de las distancias de Itakura-Saito y COSH los resultados obtenidos son malos, ya que los valores aumentan, incluso llegando a tener S_{B0} como el valor mayor. Esto es inesperado ya que estas distancias se habían desempeñado bien anteriormente.

Por último, la distancia WLR obtuvo muy buenos resultados, donde los valores tienen van disminuyendo marcadamente entre cada señal S_B, con y sin threshold. Esta distancia no se había desempeñado satisfactoriamente antes, pero funciona bien con señales donde cambia la envolvente.

En cuanto a rendimiento, se confirman lo resultados anteriores, donde la función Pointwise es la más eficiente mientras que el cálculo de las características es la más costosa. El tipo de señales no afecta el tiempo de procesamiento, siempre y cuando se preserve la frecuencia de muestreo y duración.

Las distancias que mejor se desempeñaron en este experimento fueron la WEDM y la WLR, donde el costo de procesamiento es casi 10 mayor para la WLR.

Los espectrogramas mostrados en las figuras 5.5 y 5.6 indican un caso ligeramente que en los experimentos anteriores. Mientras que S_O y S_A presentan claramente las secciones de la envolvente, donde es visible las zonas donde cambia la intensidad de las componentes, esto no sucede tan evidentemente en las señales de S_B. En estas últimas, la envolvente si se aplica y se ve reflejada en las componentes de las senoidales principales, sin embargo, los artefactos adicionales generados por los errores de redondeo están presentes durante toda la duración de la señal. El nivel de estos artefactos es muy bajo y no afecta la percepción pero es suficientemente notorio como para identificarse en los espectrogramas.

5.1.4. General

El uso de threshold no ayuda a ninguna función de distancia. En el mejor de los casos los valores obtenidos son similares a aquellos sin threshold, pero el costo de procesamiento es mucho mayor. Por lo tanto, no se usará el threshold en ninguno de los siguientes experimentos.

En general, las funciones de WEDM e Itakura-Saito se comportaron bien a lo largo de todos los experimentos, pero es evidente que todas las distancias reaccionan de diferente manera dependiendo de la naturaleza de las señales analizadas. Debido a que la la distancia WEDM se ejecuta más rápido que Itakura-Saito, se utilizará esta

Función	S_{B3}	S_{B2}	S_{B1}	S_{B0}	S_A	S_{noise}	Tiempo
Pointwise	0.023,613	0.009,171,7	0.002,960,5	0.002,451,3	$1.108{,}7\times10^{-13}$	0.104,59	1.390,7
DFT	0.036,134	0.003,623,9	0.001,678,1	0.000,901,51	$6.015{,}4\times10^{-14}$	0.341,3	3.237,4
Características	0.616,72	15.215	47.465	42.975	$1.408{,}7\times10^{-09}$	6,561.6	120.76
Dif. Cuadrado (Thld.)	$5.125{,}6\times10^{-06}$	$1.170{,}8\times10^{-06}$	6.102×10^{-08}	$3.831{,}8\times10^{-08}$	$1.613{,}6\times10^{-30}$	$1.120{,}1\times10^{-05}$	37.024
Dif. Cuadrado	$5.125{,}6\times10^{-06}$	$1.170{,}8\times10^{-06}$	6.102×10^{-08}	$3.831{,}8\times10^{-08}$	$1.613{,}6\times10^{-30}$	$1.120{,}1\times10^{-05}$	2.120,2
WEDM (Thld.)	$1.525{,}7\times10^{+10}$	4,197.5	3,352.9	3,214.6	$5.155{,}7\times10^{-25}$	$8.713{,}6\times10^{+17}$	31.134
WEDM	501.76	0.038,983	0.006,610,3	0.006,135,5	$1.080{,}6\times10^{-24}$	$1.764{,}1\times10^{+11}$	2.632,6
Itakura-Saito (Thld.)	$6.524\times10^{+25}$	$2.946{,}5\times10^{+25}$	$3.377{,}9\times10^{+25}$	$2.309\times10^{+25}$	$4.551{,}9\times10^{-14}$	$4.067{,}3\times10^{+10}$	54.212
Itakura-Saito	$6.463{,}3\times10^{+07}$	$6.831{,}1\times10^{+07}$	$5.030{,}6\times10^{+07}$	$1.985{,}2\times10^{+09}$	$9.641{,}5\times10^{-10}$	$4.067{,}1\times10^{+10}$	24.144
COSH (Thld.)	$2.328{,}8\times10^{+17}$	$5.812{,}4\times10^{+14}$	$3.343{,}2\times10^{+14}$	$3.198\times10^{+14}$	$1.332{,}3\times10^{-15}$	$3.085{,}8\times10^{+22}$	39.284
COSH	$5.769{,}7\times10^{+08}$	$4.631{,}4\times10^{+08}$	$4.807{,}8\times10^{+08}$	$4.952{,}4\times10^{+08}$	$2.406{,}8\times10^{-10}$	$7.315{,}9\times10^{+15}$	4.979,5
WLR (Thld.)	0.321,99	0.026,207	0.002,309,1	0.001,481,1	$5.155{,}7\times10^{-25}$	14.941	56.968
WLR	0.321,94	0.026,206	0.002,309,1	0.001,481	$1.080{,}6\times10^{-24}$	8.743,6	21.491
LOG (Thld.)	$1.725{,}8\times10^{+07}$	$6.383{,}5\times10^{+06}$	$2.552{,}4\times10^{+06}$	$2.362{,}3\times10^{+06}$	0.000,890,4	$1.219\times10^{+08}$	55.391
LOG	$7.616{,}4\times10^{+06}$	$4.541\times10^{+06}$	$3.701{,}5\times10^{+06}$	$3.603{,}5\times10^{+06}$	0.081,212	$4.726{,}5\times10^{+07}$	22.084

Table 5.3: Resultados para la tercera señal de prueba. - Se muestran las diferencias de *fitness* entre la señal S_O y la señal indicada en cada columna, además del tiempo de ejecución. Las cantidades son sólo para comparación y no representan ninguna unidad física.

distancia para los cálculos de *fitness* de los siguientes experimentos, y sólo se utilizará otra distancia donde se indique.

También cabe destacar que el cálculo de la envolvente ofrece la mayor dificultad, donde todas las funciones de distancia presentan cambios más pequeños o comportamiento incorrecto. Además de que no se esperaba encontrar los valores exactos de los parámetros de la envolvente.

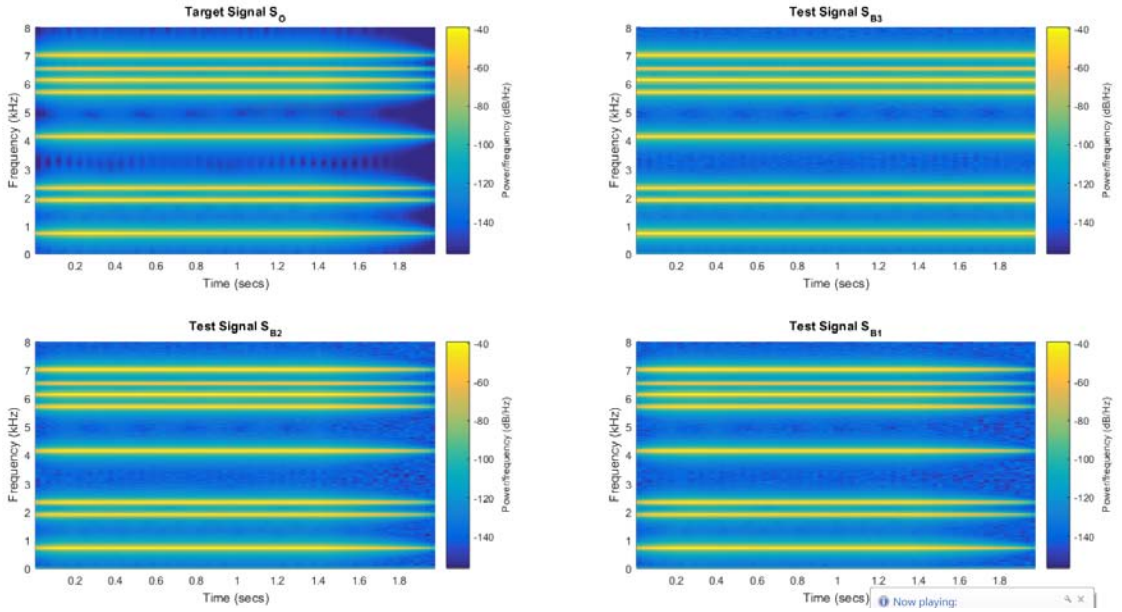

Figure 5.5: Evaluación de *fitness* - Espectrogramas para tercera señal de prueba.

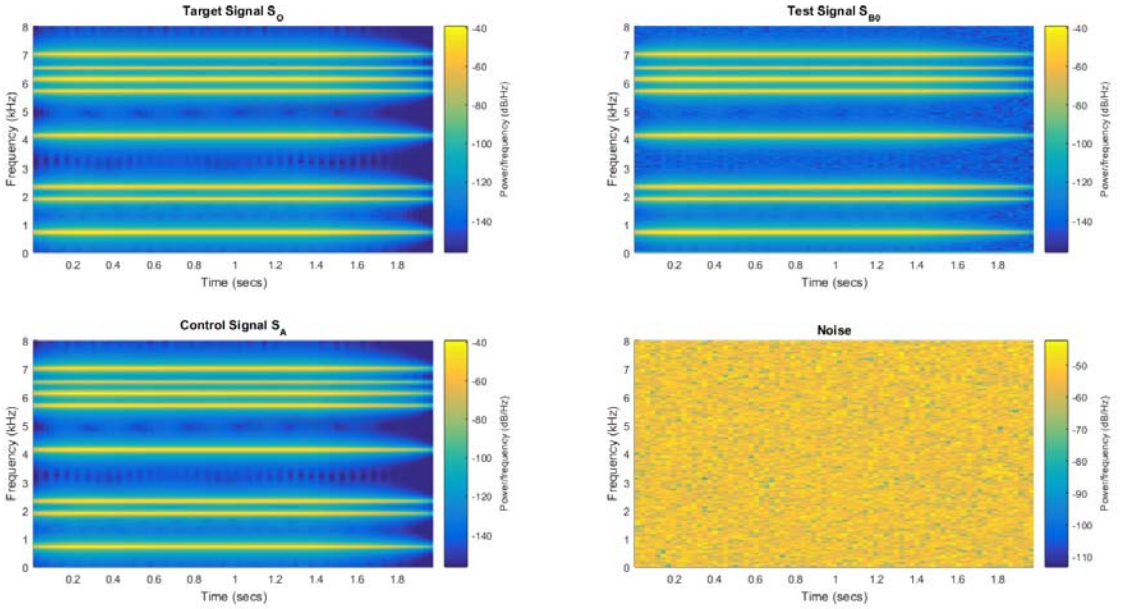

Figure 5.6: Evaluación de *fitness* - Espectrogramas para tercera señal de prueba.

5.2. Calibración de *Audiovolution*

En la tabla 5.4 se muestran los resultados promedio para la evolución de cada una de las señalas de este experimento. Se muestra el valor final de *fitness*, la cantidad de generaciones ó ciclos, la cantidad de individuos seleccionados (casos donde el clon mutado fue mejor que el padre) y el tiempo que operó el experimento. Para poder analizar estos resultados es importante considerar la condición de salida del algoritmo. Primero, se buscó que la percepción subjetiva del sonido resultado fuera similar al sonido objetivo sin importar los valores exactos de los parámetros. Para evaluar esta percepción, se comparó cada sonido y se clasificó en 3 posibles categorías: *buena* cuando son muy similares, *regular* cuando hay similitud pero hay elementos faltantes o notablemente distintos, y *mala* cuando hay poca similitud.

En caso de no generar un sonido similar, la segunda condición de salida ocurre cuando la evolución se ha estabilizado, es decir, que no haya seleccionado nuevos individuos durante 1 minuto. Esto no quiere decir que la evolución haya fallado, sino que se requeriría mayor tiempo para obtener conclusiones definitivas.

En general, para todas las señales que tiene envolvente de amplitud, la evolución no obtiene los valores específicos configurados. No obstante, la percepción subjetiva de los sonidos es muy cercana. Esto soporta los resultados obtenidos en el experimento de *fitness* anterior, donde la diferencia de distancias entre señales con valores cercanos de envolvente es muy pequeña, y difícilmente el algoritmo va a tomar los valores exactos en poco tiempo de evolución. Por otro lado, cuando se llega a un sonido similar, la frecuencia fundamental es algo que se obtiene temprano en la evolución, dado que tiene gran influencia sobre el cálculo de *fitness* (ver figura 5.9).

Señal	Generación	Seleccionados	Fitness	Tiempo	Similitud
S_{C0}	5,725	23	0.875929801	00:48	Buena
S_{C1}	8,760	43	1.14533053552321	01:20	Buena
S_{C2}	21,450	29	6.38579246902101	04:50	Buena
S_{C3}	10,944	20	0.03963814223026	02:37	Buena
S_{C4}	19,662	50	1.26456072403903	03:22	Buena
S_{C5}	32,753	77	0.164837732276372	05:26	Regular
S_{C6}	60,021	34	2427.16013395195	07:02	Mala
S_{C7}	116,275	38	1.31489990115801	35:00	Mala

Table 5.4: Resultados para la calibración de *Audiovolution*.

Las primeras señales se desempeñan bien. Cuando la señal consta de una sola senoidal, como es el caso de S_{C0}, S_{C1}, el algoritmo rápidamente identifica la frecuencia y se obtiene un sonido muy parecido en poco tiempo de ejecución y pocos individuos seleccionados. En cambio, cuando la primitiva tiene otra forma de onda, el algoritmo es menos preciso, pero el sonido resultado sigue siendo bueno. Para S_{C2} y S_{C3}, hay casos donde el algoritmo obtiene primitivas adicionales con baja amplitud o con frecuencias cercanas a armónicos presentes en la señal original. Estás primitivas adicionales no afectan la percepción del sonido, aunque la composición de la señal resultante ya es distinta. En otros casos en vez de obtener la frecuencia fundamental exacta, obtiene una sub-armónica, por ejemplo 226 Hz en vez de 678. Del mismo modo parámetros como el ciclo de trabajo no son obtenidos con precisión, pero su contribución a la percepción

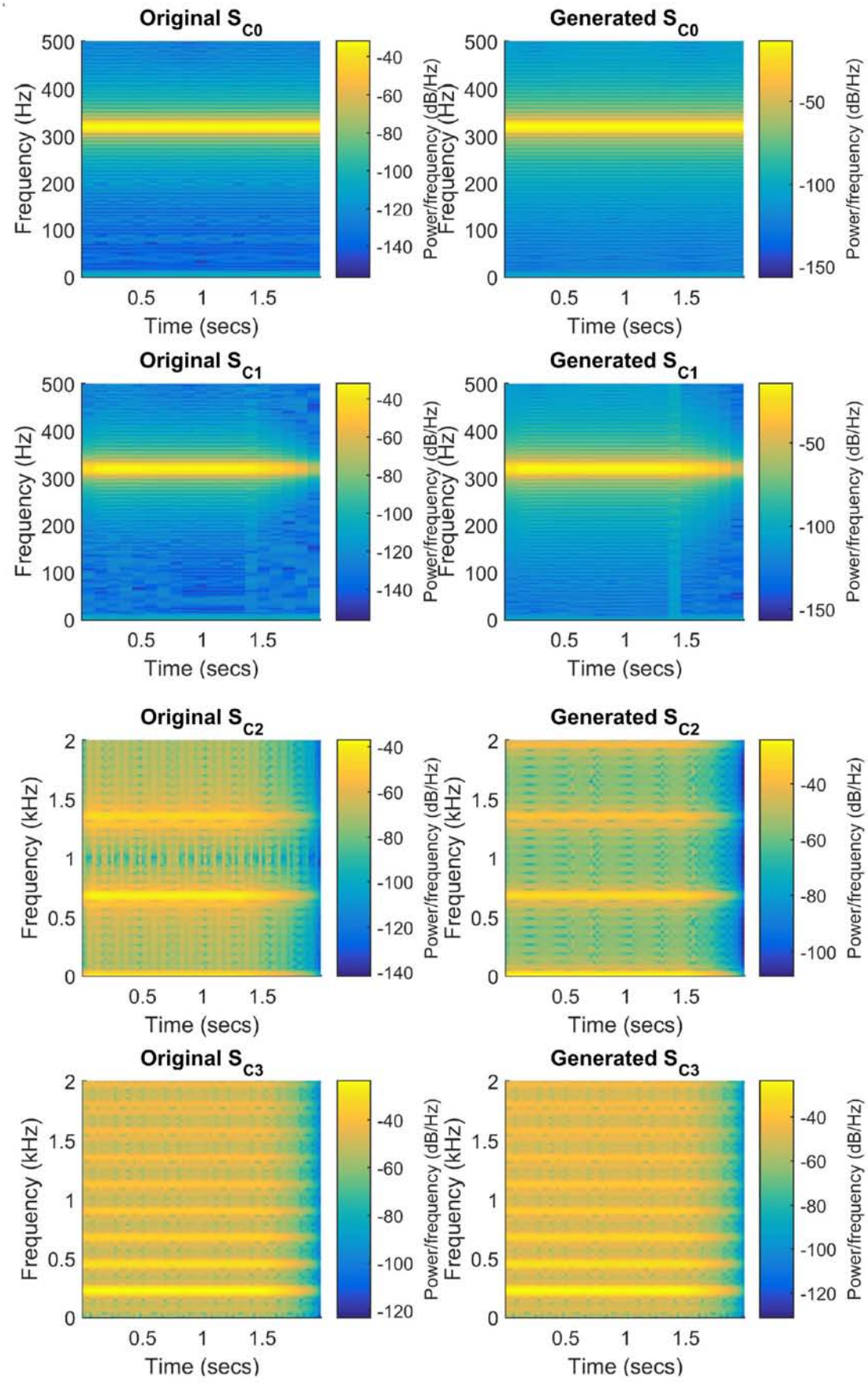

Figure 5.7: Calibración - Espectrogramas para las primeras 4 señales de prueba.

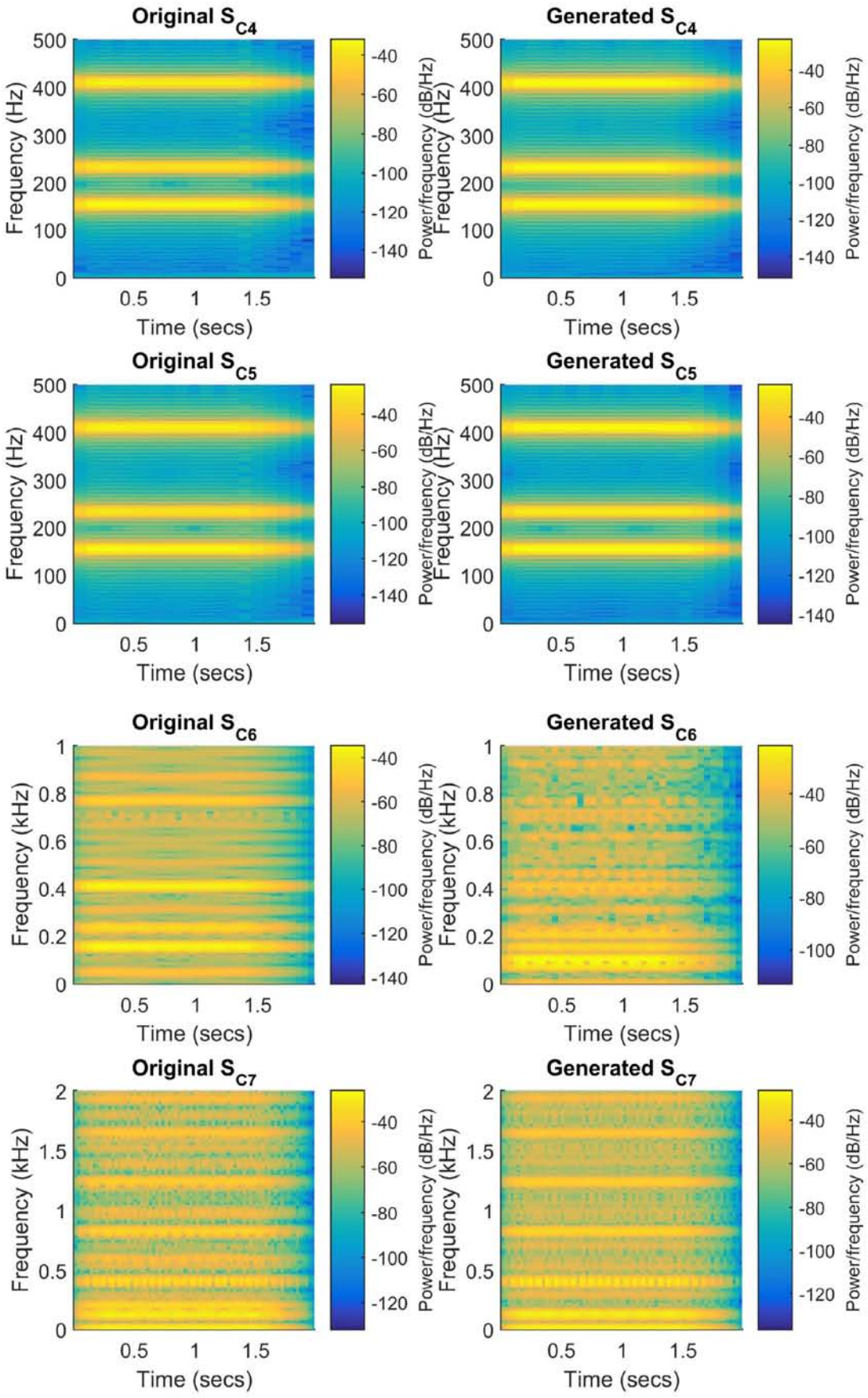

Figure 5.8: Calibración - Espectrogramas para las siguientes 4 señales de prueba.

del sonido es pequeña.

Cuando la señal esta compuesta de variables primitivas, el algoritmo obtiene buenos resultados siempre y cuando no se usen formas de onda complejas. Para S_{C4} y S_{C5}, la evolución es más larga que en las señales anteriores llegando a sonidos muy parecidos al objetivo. Los parámetros de fase no fueron obtenidos con exactitud, debido a que lo importante no es la fase individual de cada primitiva, sino la combinación de todas en la señal total.

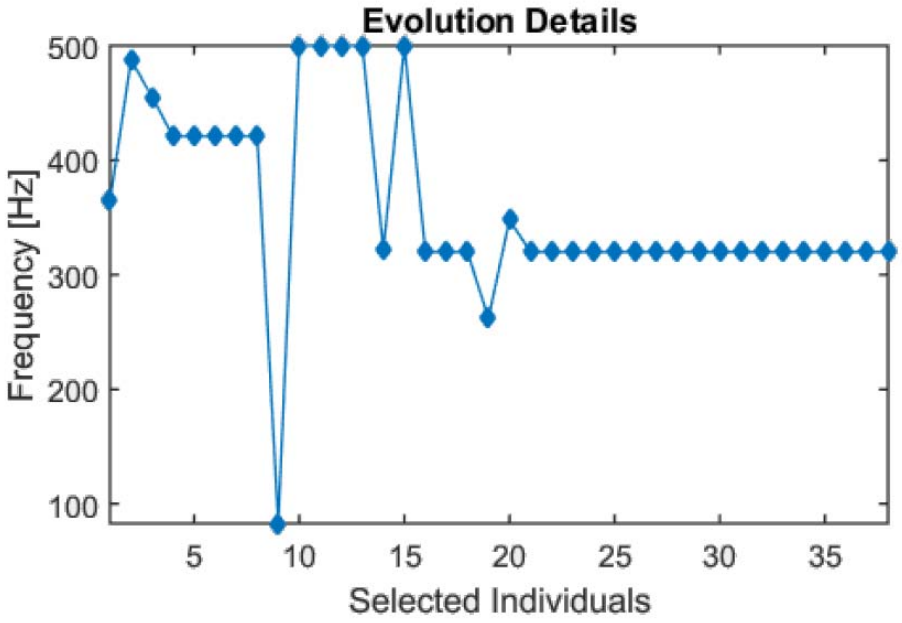

Figure 5.9: Evolución de frecuencia para S_{C1} - Se muestra cómo se fue modificando la frecuencia para la señal S_{C1}, donde se estabiliza en 320 Hz después de 20 individuos seleccionados.

Finalmente, para S_{C6} y S_{C7} que tienen múltiples primitivas con formas de onda complejas, es evidente que el algoritmo queda limitado. Las evoluciones toman considerablemente más tiempo que las anteriores, sin embargo, la cantidad de individuos seleccionados no aumenta. Esto indica que la evolución toma más ciclos para encontrar individuos que mejoren la solución, y dado que cada ciclo utiliza una mutación, esto muestra que la cantidad de mutaciones descartadas es mayor. A pesar de que S_{C7} tiene un valor final de *fitness* similar a las señales anteriores, el sonido resultado tiene poco parecido al sonido objetivo.

Los espectrogramas de las señales de este experimento se observan en las figuras 5.7 y 5.8. Tanto para S_{C0} y S_{C1}, los resultados son prácticamente idénticos a los originales. Se identifican claramente la componente principal y hay poco esparcimiento a frecuencias vecinas. Hay unos artefactos presentes pero son a niveles muy bajos, de -120 dB por lo que no afectan la percepción. Para S_{C2} y S_{C3} las existen diferencias sutiles pero importantes. Las componentes principales están bien identificadas, pero las armónicas tienen menos intensidad en los espectrogramas generados.

Para S_{C4} y S_{C5}, los espectrogramas también son muy parecidos, salvo por unas pequeñas diferencias en las zonas marcadas por la envolvente. En las originales estas transiciones son más suaves. Para S_{C6} y S_{C7} se aprecia una diferencia considerable. En

S_{C6}, el espectrograma original muestra claramente las componentes principales y una variedad de armónicas con distintos niveles. Por otro lado, el espectrograma generado no tiene tal claridad. No están presentes todas las componentes principales y hay fluctuaciones a lo largo del tiempo que alteran el nivel de los armónicos. En S_{C7} la cantidad de armónicos visibles no son iguales que el original.

Hay ocasiones donde un individuo genera dos primitivas *DNAwave* con la misma frecuencia fundamental y forma de onda. Al momento de sintetizar la señal, esto equivale a un cambio en la amplitud para esa frecuencia, pero no agrega más variación al espectro. Esto es un problema ya que estos individuos están aumentando la complejidad del ADN (a mayor número de primitivas, mayor número de cálculos necesarios) sin obtener un beneficio real. Una posible solución es modificar el proceso de mutación para que los individuos no generen primitivas con la misma frecuencia de otra primitiva, sin embargo, este proceso aumentaría la complejidad y tiempo de ejecución del ciclo. Dado que esta situación no se presentó en gran parte de los casos ejecutados, no se implementaron modificaciones.

Es posible que los resultados del experimento mejoren si se utilizan evoluciones más largas, o si se modifican los parámetros de evolución para restringir aún más los valores posibles para la mutación (por ejemplo, usar sólo formas de onda triangulares si se sabe que la señal objetivo es una onda triangular). No obstante, estos experimentos demuestran las limitantes del proceso en general.

5.3. Replicación de sonidos - Sintetizador Musical Simple

Los espectrogramas obtenidos de este experimento se muestran la figura 5.10, mientras que las señales en el tiempo se observan en la figura 5.11. Los resultados de la evolución están disponibles en la tabla 5.5. Los datos desplegados son los mismos que en el experimento anterior, usando el promedio de los indicadores principales de la evolución para cada señal, además de un calificador subjetivo de la percepción de la señal generada.

A pesar de que las señales de prueba de este experimento son simples porque se componen una sola primitiva, se encontró que el algoritmo no alcanza buenos resultados. Cómo primera observación, se observó que para este experimento la función de *fitness* WEDM que había sido utilizada previamente no se desempeñó satisfactoriamente. Al utilizar está función, el algoritmo tendía a generar primitivas adicionales las cuales si bien complementaban el espectro, perjudicaban notablemente la percepción de los sonidos obtenidos. Al utilizar la distancia de diferencia al cuadrado se obtuvieron sonidos mucho más similares a los de prueba, aún cuando los espectrogramas no son tan parecidos. Por esta razón, los resultados mostrados para esta sección utilizan la distancia de diferencia al cuadrado.

Señal	Generación	Seleccionados	Fitness	Tiempo	Similitud
S_{D0}	31,382	21	$5.945{,}638 \times 10^{-4}$	06:53	Regular
S_{D1}	25,172	25	$1.625{,}388 \times 10^{-4}$	05:47	Regular
S_{D2}	22,402	33	$1.199{,}413 \times 10^{-5}$	06:29	Mala

Table 5.5: Resultados para la replicación de sintetizador musical simple.

La primera señal S_{D0} es la más sencilla y consta de 1 sola senoidal, sin embargo, al

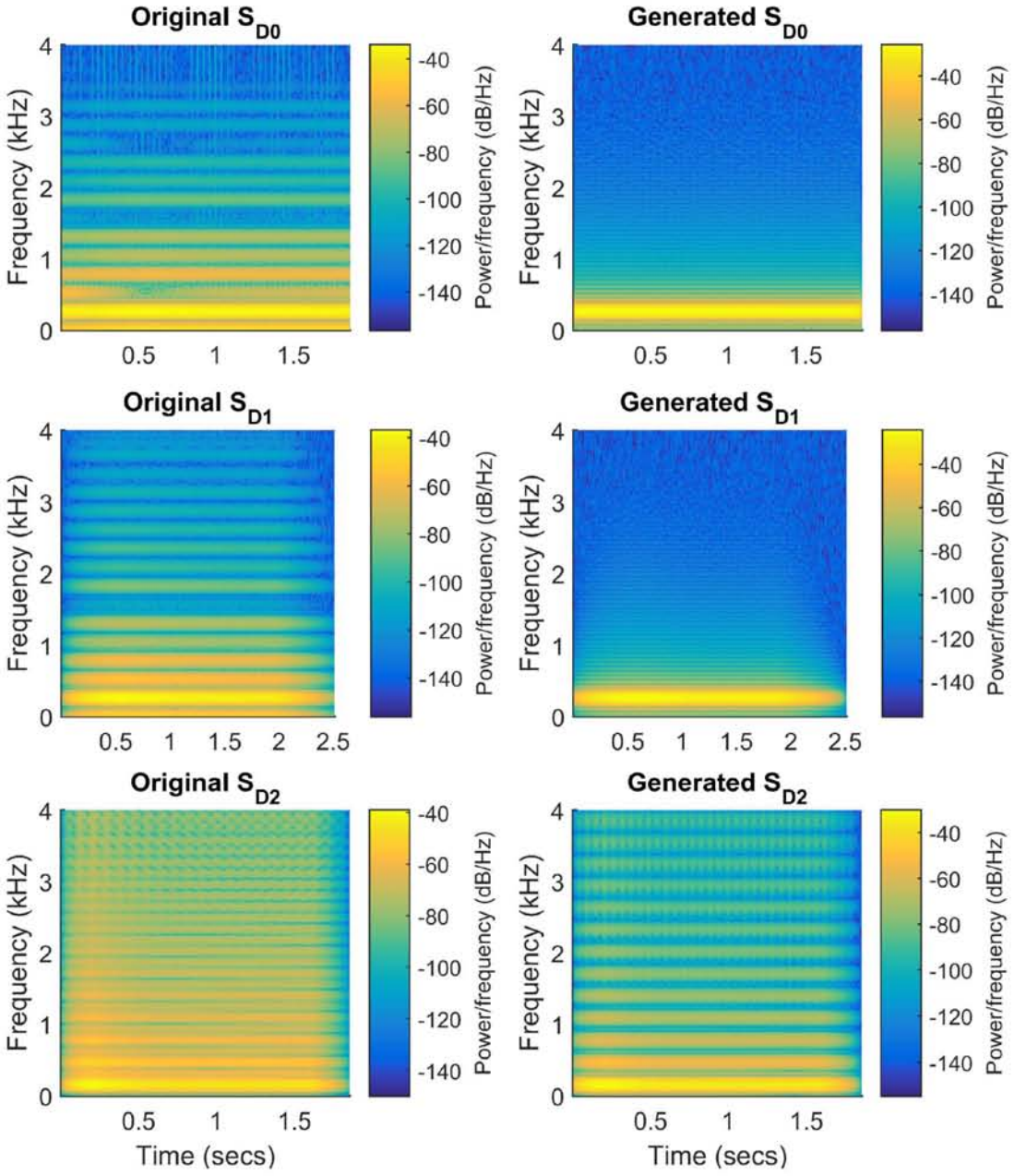

Figure 5.10: Replicación de Sintetizador Simple - Espectrogramas para las señales de prueba.

analizar el espectrograma es evidente que hay gran cantidad de armónicas y energía en otras frecuencias. El nivel de la mayoría de estás armónicas está por debajo de los -80 dB, por lo que su influencia en la percepción del sonido es limitada mas no despreciable. Por otro lado, existe unas pocas armónicas que tienen nivel considerable, llegando a -40 dB. Adicionalmente, a pesar de que la señal no fue generada con envolvente, se aprecia que la duración de algunas de estas armónicas no es constante a lo largo de toda la señal. Al utilizar un sintetizador musical en lugar de funciones matemáticas simples, es de esperarse que se encuentren este tipo de artefactos o información adicional en las señales, dado que enriquecen los sonidos generados y son preferidos por músicos y el público en general.

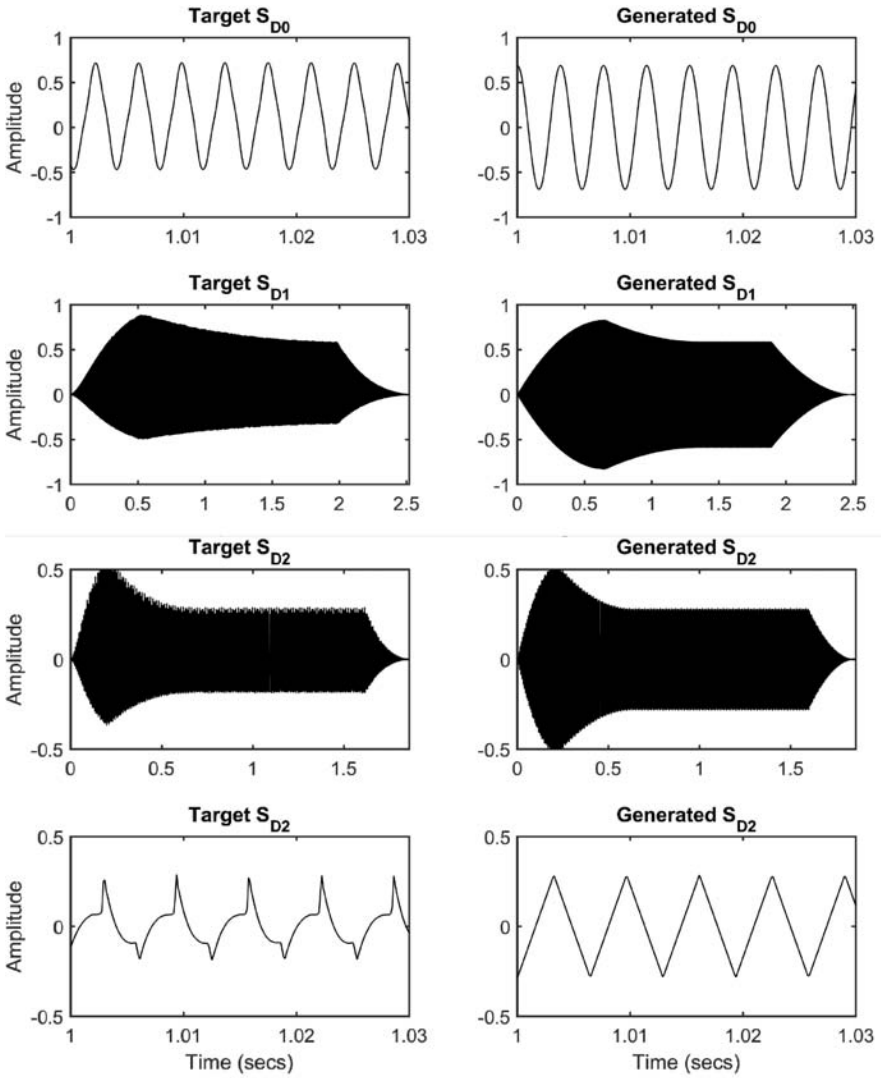

Figure 5.11: Replicación de Sintetizador Simple - Señales en el tiempo. - Para la señal S_{D0} se muestra un pequeño intervalo de tiempo, mientras que para S_{D1} se muestra completa para apreciar la envolvente de amplitud. Para S_{D2} se muestran ambos casos.

Los resultados obtenidos para S_{D0} muestran que a pesar de que la evolución operó por un tiempo mayor a experimentos anteriores, la cantidad de individuos seleccionados es pequeña. Esto porque la síntesis empleada está limitada, y aunque se generen gran

cantidad de individuos ninguno puede realmente alcanzar los artefactos existentes en la señal objetivo. El algoritmo obtuvo correctamente la primitiva base y la frecuencia fundamental sin ninguna primitiva extra. La percepción del sonido resultante es regular, ya que aunque es claro que tanto la objetivo como la generada tienen carácter de onda senoidal, la primera tiene un sonido más rico, sonando incluso como una onda cuadrada.

La señal S_{D1} es idéntica a S_{D1} salvo que se le agrega envolvente de amplitud. Para este caso, los resultados obtenidos fueron muy similares a la señal anterior, tanto los indicadores de la evolución como la calidad del sonido generado. A grandes rasgos, la envolventes generada tiene gran parecido con la señal objetivo, pero se aprecia que las curvas para la sección de ataque y decay son ligeramente diferentes. Es posible modificar la síntesis utilizada en el algoritmo para utilizar curvas diferentes para estas secciones de la envolvente. Sin embargo, dado que cada sintetizador musical emplea una implementación propia de la envolvente ADSR, y que la contribución a la percepción de sonidos de la forma de estas curvas no es de gran importancia, no es prioritario hacer esta modificación.

Por último, la señal S_{D2} utiliza una forma de onda de diente de sierra y confirma el hecho de que los sintetizadores musicales emplean elementos mucho más complejos que primitivas simples. Al analizar un periodo de la señal (ver figura 5.11) se aprecia que la forma de onda real no es diente de sierra, sino algo más elaborado. Curiosamente, el sonido resultado generado utiliza la forma de onda triangular en vez de la diente de sierra. En cuanto a la percepción, el sonido objetivo tiene un carácter especial, donde se aprecia una onda base y un ruido de alta frecuencia. El sonido obtenido es similar a la onda base del objetivo, pero carece de todos los armónicos extras que dan brillo al original.

Un aspecto que se encontró en todas las señales de prueba, fue la presencia de una componente de DC, es decir que las señales generadas por el sintetizador musical no están centradas alrededor de 0. Normalmente la componente de DC introduce ruido en forma de una senoidal de muy baja frecuencia, y aunque se aprecia este fenómeno en los espectrogramas de las señales objetivo, es notable que esto no ocurrió en las señales generadas. Este ruido adicional es poco notable en la percepción de los sonidos.

5.4. Replicación de sonidos - Sonidos Complejos

Los espectrogramas obtenidos de este experimento se muestran la figura 5.13, mientras que las señales en el tiempo se observan en la figura 5.12. Los resultados de la evolución están disponibles en la tabla 5.6. Se muestran los mismos indicadores promedio que en experimentos anteriores, y para aquellas señales donde no había un resultado común entre las ejecuciones se seleccionó un resultado representativo.

Las señales empleadas en este experimento son altamente complejas, y es poco realista que un proceso de síntesis simple como el empleado en esta tesis pueda replicar estos sonidos. Aún así se encontraron algunos resultados interesantes. Todas las ejecuciones utilizaron la función de *fitness* WEDM, sin embargo, derivado de lo observado en el experimento anterior, se probó también con la diferencia al cuadrado. Las dos funciones de distancia no mostraron comportamiento marcadamente distinto, por lo que se muestran sólo los datos obtenidos usando WEDM.

Para S_{Z0} lo primero que resalta es la variedad que se presentó en las distintas ejecuciones. A diferencia de los experimentos anteriores, no había un patrón general que

Señal	Generación	Seleccionados	Fitness	Tiempo	Similitud
S_{Z0}	32,052	22	5.384,429,868	35:00	Mala
S_{Z1}	49,947	20	1.211,952,023	35:00	Mala
S_{Z2}	76,549	25	3.037,224,956	35:00	Mala
S_{Z2}*	96,139	32	4,023,539.398	35:00	Mala

Table 5.6: Resultados para la replicación de sonidos complejos.

se alcanzara, y se llegaba a señales con poca similitud. En ocasiones se obtenían señales a base de ondas de diente de sierra, en otras se obtenían principalmente triangulares. Además, la frecuencia fundamental no fue encontrada constantemente. A pesar de la larga evolución, los individuos seleccionados son pocos, por lo que la diversidad en los sonidos generados depende fuertemente de los primeros individuos, y mutaciones subsecuentes no introducen cambios mayores. La percepción es mala, donde no es posible identificar elementos del sonido objetivo, incluso cuando a grandes rasgos los espectrogramas son similares.

La señal S_{Z1} tuvo resultados significativamente mejores que la anterior, pero siguen teniendo serias deficiencias. El espectrograma de esta señal muestra claramente como varía la intensidad de las distintas armónicas a lo largo del tiempo. En general, los armónicos superiores y las frecuencias altas tienden a desaparecer mucho más rápido que las notas bajas. Sin embargo, hay un importante aumento de frecuencias altas cerca de la mitad de la duración, propio de los instrumentos de cuerda. Esta señal tiene además una modulación de frecuencia que actúa sobre la parte final del sonido. Al igual que S_{Z0}, las evoluciones presentan pocos individuos seleccionados, aunque en este caso las ejecuciones tienden a llegar a resultados muy parecidos. Los mejores resultados generalmente utilizan una onda triangular y una diente de sierra, y encuentran correctamente la frecuencia fundamental. Al analizar la señal en el tiempo durante el estado estable, es notable que la forma de onda se asemeja bastante. Aún así, el sonido resultado es marcadamente diferente. Si hay elementos en común, pero es evidente que el timbre es muy distinto.

La señal S_{Z2} es la más compleja de todas e incluye polifonía. El espectrograma es muy similar a la señal anterior, pero hay mayor concentración de energía a lo largo de todo el espectro. La polifonía es muy evidente al escuchar el sonido, pero la notoriedad en el espectrograma depende altamente de la resolución usada. Para este caso se ejecutó un conjunto de evoluciones teniendo parámetros abiertos, y otro donde se limitaron las formas de onda para sólo utilizar senoidales (marcada como S_{Z3}* en la tabla 5.6). En ambos casos, los indicadores de evolución fueron similares, teniendo evoluciones largas pero pocos individuos seleccionados. Al usar parámetros abiertos, los sonidos generados normalmente contiene una primitiva diente de sierra de baja frecuencia junto a algunas otras triangulares de nivel mucho más bajo. La percepción de estos sonido es mala, ya que tiene virtualmente ningún parecido al objetivo y el espectrograma resultante confirma esto.

Para el caso de usar sólo senoidales, los sonidos generados se enfocan en las bajas frecuencias, obteniendo entre 4 y 8 primitivas con frecuencias menores a 1000 Hz, con un rango amplio de amplitud. No obstante, estas primitivas no corresponden a la frecuencia fundamental de ninguna de las notas del objetivo y la percepción del sonido es mala, ya que se escuchan claramente las diferentes senoidales pero no se mezclan. Del mismo modo, el espectrograma muestra poco parecido con el objetivo.

En general, la envolvente de amplitud de las señales es muy diferente a aquellas que puede generar el algoritmo. Los ataques son cortos y tienen un incremento súbito, mientras que el decay es largo y suave, además de que no restringe completamente a la señal. Hay gran cantidad de muestras que parecen no seguir los límites de la envolvente. Al ser sonidos generados con instrumentos físicos es natural que las envolventes sean mucho más complejas.

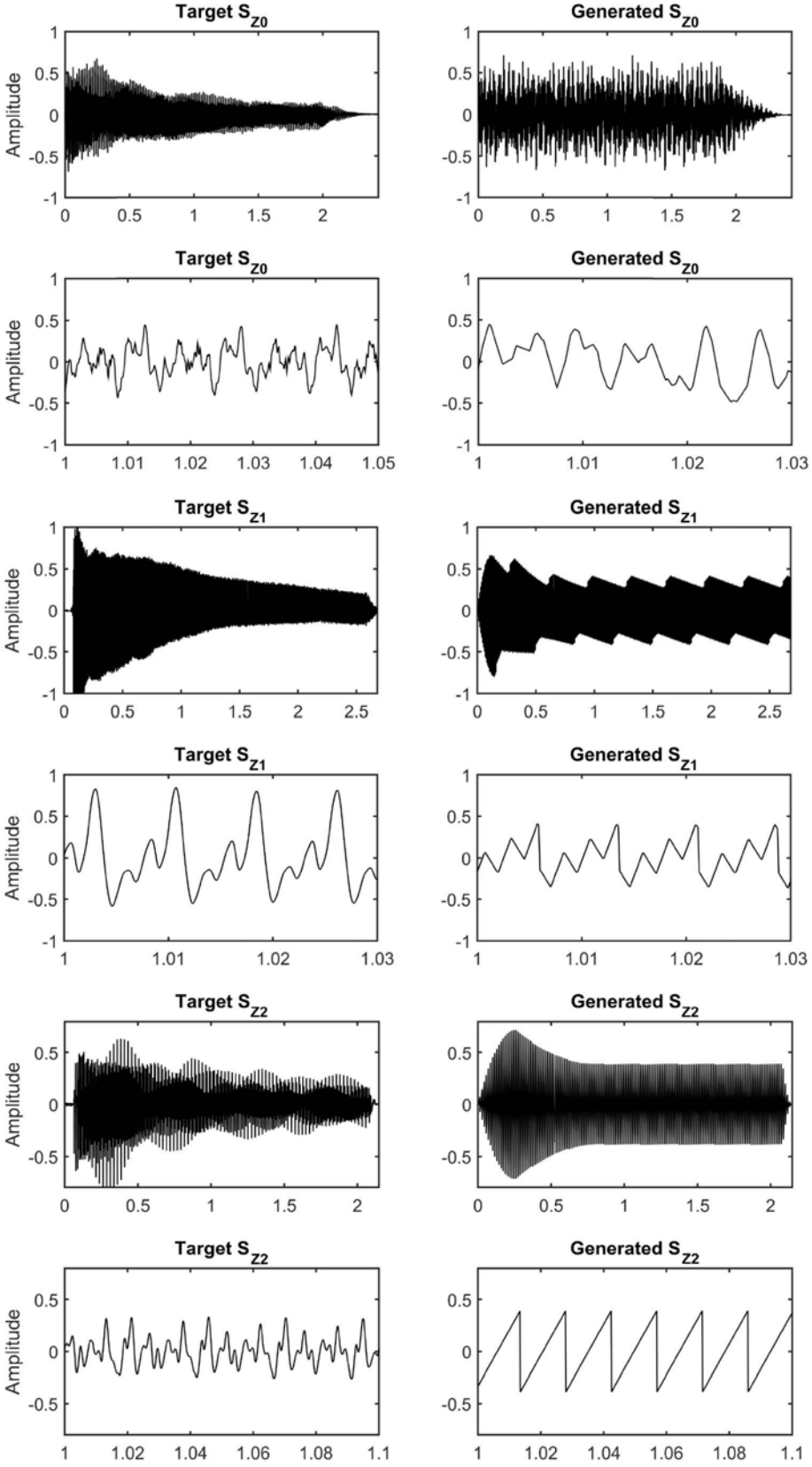

Figure 5.12: Replicación de Sonidos Complejos - Señales en el tiempo. - Para las tres señales de prueba se muestra tanto la longitud total como un pequeño intervalo de tiempo.

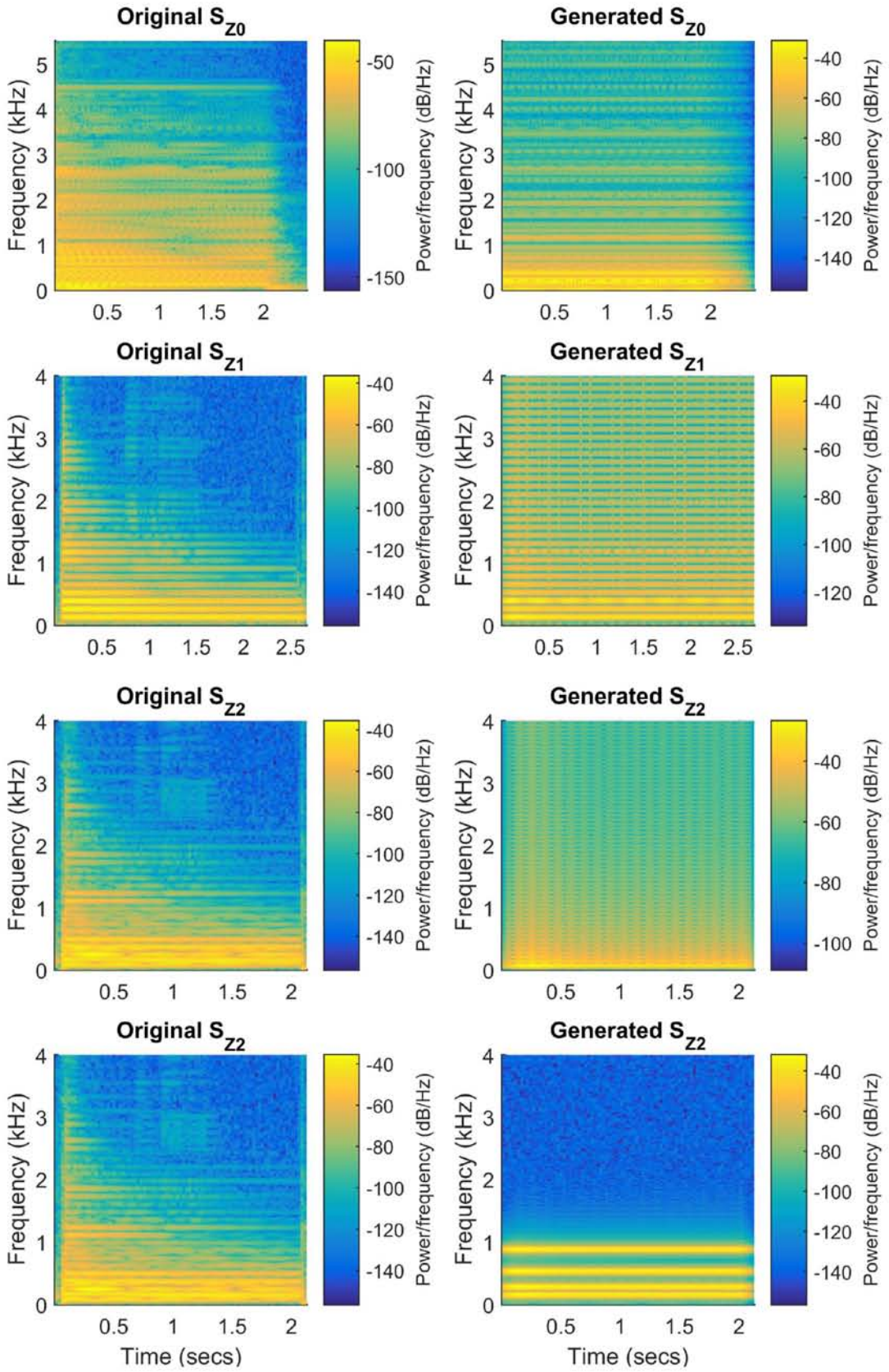

Figure 5.13: Replicación de Sonidos Complejos - Espectrogramas para las señales de prueba - Para S_{Z3} se muestra la evolución usando parámetros abiertos y aquella usando sólo senoidales.

Capítulo 6

Conclusiones

Este trabajo ha presentado resultados mixtos. Por un lado, la aplicación y biblioteca de funciones componen una plataforma robusta para realizar análisis de audio utilizando algoritmos genéticos. Por otro lado, los experimentos efectuados exponen tanto las limitantes del proyecto, como la alta complejidad que implica el estudio de estas señales utilizando cómputo evolutivo.

En primer lugar, es claro que aplicar la misma técnica de algoritmos genéticos enfocados a imágenes para señales de audio no es un proceso simple y tiene serias deficiencias. La representación de imágenes como una colección de polígonos es mucho más precisa que la representación de audio como una colección de ondas básicas. No sólo porque las imágenes son estáticas mientras que el audio depende y varía en el tiempo, sino porque la síntesis de audio es altamente complicada y no es una tarea trivial. A pesar de que los algoritmos genéticos y otras técnicas de cómputo genético han sido utilizados exitosamente en experimentos previos de audio (McDermott et al. (2008), Macret et al. (2012), Manzolli et al. (2001) entre otros), estos trabajos generalmente tienen un enfoque dirigido, limitando el tipo de señales y utilizando sonidos objetivos generados con un sintetizador determinado y probado íntegramente. Derivado de esto y de los experimentos realizados, se identifican varios componentes que podrían ser modificados y donde es necesario realizar más estudios, principalmente la síntesis, los parámetros de evolución, la función de *fitness* y la selección de señales objetivo.

La síntesis usada en esta tesis funcionó bien para señales sencillas pero queda superada por señales de audio generadas a partir de fuentes físicas. Al utilizar señales generadas por el mismo sintetizador, se obtuvieron buenos resultados y se replicaron los sonidos, confirmando la correcta operación del algoritmo genético. Además, fue posible replicar muestras de audio obtenidas a partir de un sintetizador musical (instrumento virtual), aún cuando el resultado obtenido no fue exacto. No obstante, incluso los instrumentos virtuales más simples utilizando los parámetros más básicos enriquecen el espectro de las señales generadas, y la síntesis usada falla en replicar estos detalles adicionales. Para sonidos generados a partir de fuentes físicas la síntesis no pudo aproximarse a las señales objetivo, excepto por el caso de guitarra eléctrica usando una única nota, donde aunque la percepción no fue muy cercana, la forma de onda obtenida se asemeja a la buscada.

La síntesis sí fue exitosa al replicar la envolvente general de amplitud, tanto para señales propias como aquellas generadas desde un sintetizador musical simple. A pesar de que rara vez se llegó a los valores exactos de las señales objetivo, la percepción fue muy buena y en general se aprecian todas las zonas importantes. Esto no fue el

caso para los sonidos generados a partir de fuentes físicas, donde las envolventes son más arbitrarias y no es posible representarlas con apenas 4 curvas. Para siguientes trabajos, sería conveniente medir el desempeño de mutaciones donde cada parámetro de la envolvente evoluciona por separado, e incluso utilizar une envolvente por cada primitiva generada. Esto haría posible replicar sonidos a partir de fuentes físicas, que tienen espectros dinámicos complejos.

El proceso de evolución fue exitoso en mayor parte, donde los casos fallados corresponden mayormente a deficiencias de la síntesis y no de la implementación del algoritmo genético. De todos los experimentos efectuados se obtuvieron indicadores similares independientemente de la calidad del resultado lo que indica que el proceso es consistente, y la representación del ADN de la solución es compacta y concorde a la síntesis. Sin embargo, se podría explorar más a detalle la influencia de los parámetros evolutivos sobre señales de audio, modificando el tamaño de la población, y los procesos de combinación y selección de individuos. También se podrían agregar restricciones a los procesos de mutación, siguiendo fenómenos acústicos. Por ejemplo, se podría asignar preferencias para que la síntesis tienda a usar formas de onda más sencillas, o validaciones para evitar repetir frecuencias, ya sean fundamentales o armónicas ya presentes en el individuo. Cualquier modificación realizada sobre las operaciones de mutación deberá considerar el costo computacional y evaluar si es compensado por mejores resultados.

Pese a que la comparación de las diferentes funciones de distancia propuestas para el cálculo de *fitness* presentó buenos resultados, se encontró en los experimentos posteriores que la selección de la está función influye notablemente en el éxito de la evolución y en la composición de los sonidos obtenidos. No basta con determinar una función que siga la trayectoria buscada, y es necesario considerar la naturaleza de los sonidos a replicar. La función de distancia WEDM fue la que mejor se desempeñó a lo largo de todas señales de prueba al evaluar las funciones, pero ésta no fue utilizada para replicar sonidos de un sintetizador musical simple, dado que la función de diferencia al cuadrado dio mejores resultados. Determinar la similitud entre dos señales de audio no es una tarea trivial, y en trabajos futuros se podría analizar a fondo estas y otras funciones para encontrar una medida confiable y eficaz que aplique para sonidos cortos, basadas en la percepción de timbre.

Uno de los resultados menos esperados fue que la función de distancia con base de características no sólo no se comportó satisfactoriamente, sino que incluso llegó a tener el comportamiento contrario al deseado. Al analizar los resultados del experimento de evaluación de *fitness* se aprecia que las señales control si tuvieron resultados correctos para todas las pruebas. La distancia es pequeña para S_A que tiene los mismos parámetros y grande para S_{noise} que es ruido. Son las señales con parámetros cercanos las que fallaron. Existen varias razones que pueden explicar por qué sucedió esto para el experimento en general, y es necesario realizar investigaciones posteriores para identificarlas claramente.

En primer lugar, las características seleccionadas si bien son conocidas y han sido utilizadas en experimentos anteriores, no han sido probadas específicamente para identificar similitud en el timbre de señales de audio con la naturaleza usada en esta tesis. Se sabe que individualmente cada una de las características empleadas influyen en el timbre en general, pero no qué tanto en específico. Por ejemplo, los coeficientes MFCCs son ampliamente utilizados en el análisis de señales de voz, y recientemente se han utilizado para estudiar señales de música, pero existe poco trabajo aplicado específicamente a señales cortas de audio como las utilizadas en esta tesis. Asimismo, siguiendo

el estándar se usaron los primeros 13 coeficientes, pero este número no es definitivo y bien podría hacerse un estudio detallado para identificar la relevancia de cada uno de los coeficientes actuando sobre este tipo de señales. Para otras, es necesario determinar con precisión si su contribución a la representación del timbre es significativa, de lo contrario eliminarlas, o incluso reemplazarlas por otras características más refinadas. El trabajo para definir y evaluar la importancia de características de audio para un tipo de señales es complejo y abre la puertas a gran cantidad de posibles trabajos futuros.

Otro punto importante de las características que podría explorarse más, es las construcción del vector que representa la señal y determina la distancia entre vectores. Para esta tesis se construyó un vector tomando los valores totales de aquellas características que regresan un valor único, y la media y desviación estándar para aquellas que varían por trama. Sería conveniente analizar otras posibles representaciones, quizás obteniendo un vector completo por cada trama para después realizar operaciones estadísticas más elaboradas. Adicionalmente, los vectores utilizados no aplican ningún tipo de ponderación, ya que cada valor contribuye de la misma manera a la distancia final. Al analizar con mayor profundidad la contribución de cada característica a la percepción del timbre sería posible determinar pesos para cada uno de las componentes del vector. También se podría explorar la forma en que se calcula la distancia entre vectores de dos señales. Dado que algunas características representan cantidades físicas, es posible que distancias más complejas o un cambio a forma logarítmica genere una distancia entre vectores más exacta.

Finalmente, uno de los principales componentes de este experimento es determinar que constituye una señal cercana. Para las señales S_B se modificaron los parámetros siguiendo una trayectoria lógica, donde se acercaba la frecuencia, la cantidad de primitivas, o valores de la envolventes. Sin embargo, es posible que el oído humano no siga esta lógica y es necesario explorar en primer lugar si existe el concepto de cercanía para señales de audio cortas, y en segundo cómo se determina objetivamente. Es más, quizás el concepto de cercanía no es universal para todos los escuchas. Por ejemplo, el público en general podría identificar sonidos simultáneos con frecuencias muy próximas como sonidos cercanos, pero músicos entrenados podrían clarificarlos como distintos debido a que suenan desafinados y causan molestia.

No obstante, es fundamental recordar que el objetivo del experimento fue determinar una función de *fitness* para ser utilizada dentro del algoritmo genético, por lo que cualquier estudio sobre características deberá tener presente el costo de procesamiento. Además, será necesario evaluarla la función dentro del algoritmo, ya que no hay garantía de cómo se comporte el proceso de evolución, donde por ejemplos para las señales del sintetizador musical simple se encontraron mejores resultados utilizando una señal de distancia que no había la mejor en el experimento de evaluación de *fitness*.

Los algoritmos genéticos son sin duda una herramienta poderosa para encontrar soluciones en espacios con gran cantidad de variables, pero no una arma mágica que resuelven todos los problemas. Dependen enormemente de la correcta asignación de parámetros que reduzcan el número de mutaciones irrelevantes, y sobre todo, dependen completamente de una función de *fitness* fuerte, que sea capaz de guiar la evolución. Para el caso de señales de audio, la representación del ADN de los individuos es determinada por el sintetizador, mientras que la función de *fitness* se basa en mediciones de similitud entre sonidos. Estos dos puntos quedan abiertos, y es necesario estudiarlos más exhaustivamente.

Bibliografía

Chu, W. C. (2003). *Speech Coding Algorithms: Foundations and Evolution of Standarized Coders*. John Wiley & Sons, first edition. 29

Everest, F. A. (1987a). Fundamentals of sound. In Ballou, G., editor, *Handbook for Sound Engineers: The new audio Ecyclopedia*, chapter 1, pages 3–22. Howard W. Sans Company. 10

Everest, F. A. (1987b). Psychoacoustics. In Ballou, G., editor, *Handbook for Sound Engineers: The new audio Ecyclopedia*, chapter 2, pages 23–40. Howard W. Sans Company. 10, 11, 12

FabFilter Software Instruments (2016). Fabfilter twin 2 manual. Manual disponible en: http://www.fabfilter.com/help/fftwin2-manual.pdf, Fecha de visita: Enero 2016. 51

Giannakopoulos, T. and Pikrakis, A. (2014). *Introduction to Audio Analysis a MATLAB Approach*. Academic Press, first edition. 42, 43

Goldberg, R. and Reik, L. (2000). *A Practical Handbook of Speech Coders*. CRC Press, first edition. 45

Griffiths, I. (2012). *Programming C# 5*. O'Reilly Media, first edition. 55

Hajda, J. M. (2007). The effect of dynamic acoustical features on musical timbre. In Beauchamp, J. W., editor, *Analysis, Synthesis, and Perception of Musical Sounds: The Sound of Music*, chapter 7, pages 250–268. Springer. 39

Harris, F. J. (1978). On the use of windows for harmonic analysis with the discrete fourier transform. *Proceedings of the IEEE (Volume:66 , Issue: 1)*. 29

Hartmann, W. M. (2007). Acoustical signal processing. In Rossing, T. D., editor, *Springer Handbook of Acoustics*, chapter 14, pages 503–530. Springer. 42

Husbands, P., Copley, P., Eldridge, A., and Mandelis, J. (2007). An introduction to evolutionary computing for musicians. In Miranda, E. R., editor, *Evolutionary Computer Music*, chapter 1, pages 1–27. Springer. 14

Johanson, R. (2015). EvoLisa source code. Disponible en: https://code.google.com/p/alsing/downloads/detail?name=EvoLisaSource.zip&can=2&q=, Fecha de visita: Enero 2016. 17, 35

Loizou, P. C. (2013). *Speech Enhacement: Theory and Practice*. CRC Press, second edition. 44, 45

Macret, M., Pasquier, P., and Smyth, T. (2012). Automatic calibration of modified fm synthesis to harmonic sounds using genetic algorithms. *Proceedings of the 9th Sound and Music Computing conference (SMC 2012) Copenhagen, Denmark*. 14, 79

Manzolli, J., Maia, Jr., A., Fornari, J., and Damiani, F. (2001). The evolutionary sound synthesis method. 14, 79

Mathew, T. V. (2005). Genetic algorithm guide. 7

McDermott, J., Griffith, N., and O'Neill, M. (2008). Evolutionary computing applied to sound synthesis. In Romero, Juan J., M. P., editor, *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*, chapter 4, pages 81–101. Springer. 41, 42, 79

McDermott, J., Griffith, N. J., and O'Neill, M. (2005). Toward user-directed evolution of sound synthesis parameters. *Applications of Evolutionary Computing (Chapter)*. 41, 42

McDermott, J. M. (2008). *Evolutionary Computation Applied to the Control of Sound Synthesis*. PhD thesis, The University of Limerick. 5, 14, 37

McGuire, S. and van der Rest, N. (2016). *The musical art of synthesis*. Focal Press, first edition. 37, 40

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, third edition. 7

Misra, H., Ikbal, S., Bourlard, H., and Hermansky, H. (2004). Spectral entropy based feature for robust ASR. *Proceedings of the 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP'04, vol. 1, IEEE*. 43

Mitchell, M. (1996). *An Introduction to Genetic Algorithms (Complex Adaptive Systems)*. MIT Press, first edition. 5, 9

Müller, M., Ellis, D. P. W., Klapuri, A., and Richard, G. (2011). Signal processing for music analysis. *IEEE Journal of selected topics in signal processing*. 43

Moore, B. C. (2007). Psychoacoustics. In Rossing, T. D., editor, *Springer Handbook of Acoustics*, chapter 13, pages 459–495. Springer. 11, 12, 33, 40

Peeters, G., Giordano, B. L., Susini, P., Misdaris, N., and McAdams, S. (2011). The timbre toolbox: Extracting audio descriptors from musical signals. *The Journal of Acoustic Society of America*. 40, 42

Rocha, B., Bogaards, N., and Honingh, A. (2013). Segmentation and timbre similarity in electronic dance music. *Proceedings of the Sound and Computer Music Conference 2013*. 43

The Mathworks Inc. (2016). Envelope extraction using the analytical signal. Disponible en: http://www.mathworks.com/help/signal/ug/envelope-extraction-using-the-analytic-signal.htm, Fecha de visita: Enero 2016. 42

Weeks, M. (2011). *Digital Signal Processing: Using MATLAB and Wavelets*. Jones and Barlett Publishers, second edition. 37

XLN Audio AB (2016). Addictive keys. Manual disponible en: http://www.xlnaudio.com/addictivekeys, Fecha de visita: Enero 2016. 53

Young, E. D. (2007). Physiological acoustics. In Rossing, T. D., editor, *Springer Handbook of Acoustics*, chapter 12, pages 429–453. Springer. 10

Yu, X. and Gen, M. (2010). *Introduction to Evolutionary algorithms*. Springer, first edition. 7, 8