Emulación de estilos musicales mediante modelos de Markov controlados por ontologías pesadas

Máster en Inteligencia Artificial Avanzada
Escuela Técnica Superior de Ingeniería Informática

Autor: D. Brian Santiago Martínez Rodríguez

Directora: Dña. Ángeles Manjarrés Riesco

Resumen

La composición musical asistida por ordenador es una disciplina del conocimiento que ha utilizado desde sus inicios multitud de técnicas relativas a la Inteligencia Artificial. El diseño de modelos de Markov para la generación musical de melodías ha sido ampliamente utilizado con la finalidad de emular un determinado estilo musical. Sin embargo, un estudio detallado de los resultados obtenidos por las principales investigaciones pone de manifiesto la ausencia de direccionalidad en las composiciones generadas mediante estos modelos. Se impone la aplicación de reglas de composición o la hibridación con otros métodos para encontrar resultados musicalmente interesantes. En el presente trabajo de investigación proponemos incorporar a los modelos markovianos la definición de una ontología pesada, construida mediante las definiciones formales y restricciones armónicas utilizadas en el discurso musical, para garantizar la coherencia armónica y estructural del material musical generado, analizando de forma independiente las características armónicas y rítmicas de la música, y utilizando nuevos frameworks para la musicología asistida por ordenador, junto con técnicas y frameworks procedentes del Procesamiento del Lenguaje Natural. El modelo de lenguaje musical será entrenado bajo corpus musicales de los compositores J. S. Bach, Monteverdi, Palestrina y estilos Jazz y Folk. Se realizará una evaluación de los resultados compositivos consistente en la realización de una serie de encuestas, entre la que se encuentra un *Test de Turing*, a un grupo de expertos formado por estudiantes de distintas especialidades del Conservatorio Superior de Música. La comparación de los resultados de nuestra evaluación con otros resultados reportados en la literatura muestra una gran bondad de nuestro sistema en la emulación de estilos, especialmente en el caso del Jazz.

Summary

Computer-assisted composition is a discipline of knowledge that has used many techniques related to Artificial Intelligence since its beginning. The design of Markov models for the musical generation of melodies has been widely used in order to emulate a certain musical style. However, a detailed study of the results obtained by the main investigations reveals the absence of directionality in the compositions generated by these models. The application of composition rules or hybridization with other methods is required to find musically interesting results. In the present research work we propose to incorporate the definition of a heavy ontology into Markov models, built by means of the formal definitions and harmonic restrictions used in musical discourse to guarantee the harmonic and structural coherence of the same, analyzing in an independent way the harmonic and rhythmic music, and using new frameworks for computer-assisted musicology, along with techniques and frameworks from the Processing of Natural Language. The language model will be trained under the musical corpus of the composers J. S. Bach, Monteverdi, Palestrina and Jazz and Folk styles. An evaluation of the compositional results will be carried out, consisting in the realization of a series of surveys to a group of students of the Conservatory of Music. The comparison of the results of our evaluation with the results obtained by other articles show a great kindness of our system in the emulation of styles, especially in the case of Jazz.

Agradecimientos

El presente trabajo final de Máster constituye un símbolo: es el broche final a dos años de esfuerzo, dedicación, renuncia y sacrificio en los que he tenido que dar lo mejor de mí mismo, física y mentalmente, tanto a nivel académico como humano, artístico, docente, incluso burocrático, en muy diversas circunstancias, no sólo universitarias. Hay que decir que ésto no constituye reproche alguno: la vida, entre 2017 y 2019, me ha regalado con algunos de los mejores momentos imaginables –exponerlos en estas líneas sería insolente y poco pudoroso por mi parte–, en una suerte de clímax –mitad agónico, mitad catártico– plagado de cambios, metas conseguidas y nuevas incertidumbres a la deriva sobre el horizonte.

La redacción y posterior defensa de esta investigación completa un ciclo, constituye un punto y seguido, cierra el vórtice de la espiral que da paso a otro capítulo (aunque ciertamente siempre me haya irritado la metáfora de entender las etapas vitales como páginas de un libro que el viento va pasando).

De lo inexorable de las horas no queda otra opción sino aceptarlo y agradecer por ello, y puestos a agradecer, comencemos.

Me gustaría, en primer lugar, agradecer a mi tutora Ángeles por su constante apoyo y dedicación a lo largo de estos dos años, primero en versión tutela, luego en modalidad *full time*, sin la cual esta investigación no hubiera salido adelante. Gracias de todo corazón por ofrecerme la posibilidad de realizar este trabajo sobre *IA* y música.

Tengo que agradecer también a mi madre, mi padre, y mi tía por su apoyo constante e incondicional durante estos convulsos meses de cambios vitales y difíciles

decisiones. También a Jaume, por aguantarme, y a mi hermano, al que quiero enviar desde aquí mucho ánimo con su tesina de máster sueco sobre cetáceos, que está actualmente desarrollando en Islandia, sobreponiéndose al frío polar y tomando medidas de campo –más bien en océano, subido en un barco–. Hermanito, cuando vuelva a ser joven quiero ser como tú y además no marearme. Finaliza este párrafo con el agradecimiento a mi perra Raise, sencillamente por el simple hecho de existir: aunque nunca será capaz de entenderlo, es sin duda el mejor animal de la tierra –esperemos que ningún otro se dé por ofendido–.

Toca el turno ahora de las menciones especiales: Toni, éste es tu lugar –y eso que yo te hacía de voluntad más férrea, curtida por las oposiciones y los exámenes de Fuga, de alma dura como el grafito de los lápices de Bach (está bien, admito que ni el grafito es duro ni probablemente Bach utilizase lápices...)–, aprovecho para agradecerte los buenos momentos de risas y *Monsters* y recriminarte, con total impunidad, que tanto tú como Elena me llevaseis la corriente aquella desdichada tarde en la que, merendando en un bar del Cabañal, os conté que tenía una idea...

Mando desde mi renovada madriguera abrazos, besos y achuchones, distribuidos dionisíaca y aleatoriamente, para el resto de mis amigos y Nina, así como para sus presentes y futuros retoños: desdichadas criaturas del siglo XXI que sin duda trabajarán gustosamente para arreglar el desastre de mundo que les hemos legado y de paso, pagarán nuestras pensiones.

El presente trabajo está dedicado a la memoria a la de J. S. Bach, a la de Lejaren Hiller, a la de Giovanni Pierluigi da Palestrina, a la de mis exalumnos de la especialidad de *Jazz* (aunque algunos no estén todavía muertos), a la de Claudio Monteverdi, por supuesto a la de Andréi Andréyevich Márkov, y por último y muy especialmente a la de mis abuelos.

Mientras estés vivo, brilla, no dejes que nada te entristezca más allá de la medida porque corta es la vida por cierto, y su retribución el tiempo exige.
Epitafio de Seikilos *(ca. 1225 a.C.)*

Índice general

Índice de figuras

Índice de tablas

Glosario

Glosario de términos musicales según el Diccionario Enciclopédico de la Música:

- **Acorde**: Dos o más notas que suenan juntas.
- **Cadencia**: Movimiento armónico convencionalmente asociado con el final de una frase o sección.
- **Cantus firmus**: Melodía preexistente que se toma como base para una nueva composición.
- **Direccionalidad**: La música es el resultado de relaciones de tensión y distensión sonoras que no son estáticas sino dinámicas, ya que se producen en el tiempo y tienen una organización rítmico-periódica, relativa a funciones tonales armónicas producidas por ciertas anatomías interválicas. En consecuencia, debemos entender la música como un movimiento. La direccionalidad condiciona el grado de tensión o distensión del discurso musical.
- **Dominante**: Quinto grado de una escala. En la armonía tonal, acorde que acumula la máxima tensión.
- **Grado**: Nota de la escala musical a la que se hace referencia.
- **Forma binaria**: Estructura musical que tiene dos partes o secciones. La primera modula de la tonalidad principal y concluye con una cadencia, por lo general en la dominante. La segunda sección regresa a la tónica.

- **Forma ternaria**: Estructura musical en tres partes o secciones. La forma puede representarse con el esquema de letras ABA, donde la sección final es repetición de la primera.
- **Modulación**: Recurso mediante el que se abandona una tonalidad para entrar en otra
- **Tonalidad**: Sistema de organización de la altura de las notas en el que los elementos guardan entre sí un orden jerárquico de mayor a menor importancia.
- **Tónica**: Primer grado de una escala. En la armonía tonal, acorde que acumula la mínima tensión y máxima estabilidad.

Capítulo 1

Introducción

1.1 Antecedentes

En los siguientes apartados expondremos de manera concisa el estado actual de la composición musical asistida por ordenador, haciendo especial énfasis en la profusa utilización de técnicas pertenecientes al ámbito de la Inteligencia Artificial que se ha desarrollado desde la segunda mitad del siglo XX hasta nuestros días; exponiendo además los principales paradigmas utilizados para resolver el problema general de la creatividad artificial, en particular aplicado a la emulación de estilos musicales. Justificaremos la necesidad de la presente investigación, reseñando las novedades incluidas en ésta y presentando las motivaciones personales que nos han guiado a lo largo de la misma.

1.1.1 Contexto

El problema de la creatividad artificial aplicada a la composición musical asistida por ordenador se remonta a la década de los cincuenta, con los primeros experimentos realizados por L. Hiller que cristalizaron en la composición de la suite *Illiac*, para cuarteto de cuerda, considerada como la primera composición musical realizada mediante la ayuda de una computadora. Desde entonces, el aumento de las capacidades computacionales de los ordenadores junto con el desarrollo

de numerosas técnicas relativas a la IA han resultado en un gran número de paradigmas utilizados para afrontar este problema. Tal y como encontramos en Nierhaus (2009), los principales acercamientos utilizados en la investigación sobre composición computacional tradicionalmente han sido el uso de modelos de Markov, gramáticas generativas, algoritmos genéticos, autómatas celulares, redes neuronales, entre otras técnicas relativas al *Machine Learning*.

La composición musical asistida por ordenador comprende numerosas subcategorías de investigación entre las que se encuentran la síntesis de sonido, la composición algorítmica, la composición estocástica, la armonización y orquestación automáticas, o la emulación de estilos, entre otros (Supper, 2004). Esta última pretende emular un determinado estilo musical compositivo realizando una imitación automatizada, ya sea de un determinado período compositivo o un determinado compositor. Como veremos más adelante, algunas de las técnicas de IA más frecuentemente utilizadas para la emulación de estilos han sido los modelos probabilistas, especialmente mediante el uso de n-gramas, el uso de gramáticas formales junto con la implementación de reglas y restricciones mediante autómatas finitos. En la sección del estado de la cuestión expondremos en mayor profundidad las principales aportaciones relativas a esta disciplina realizadas en las últimas décadas.

1.1.2 Justificación

A día de hoy el problema se encuentra lejos de estar resuelto. Existen interesantes aportaciones, como la de Schulze y Van Der Merwe (2010) en la que se calculan n-gramas de distintos órdenes [1] sobre un corpus de entrenamiento para realizar a posteriori composiciones, utilizando determinadas restricciones de carácter formal. Los resultados obtenidos son prometedores, encontrando que más de un tercio de los encuestados confundía las composiciones generadas por el ordenador con las generadas por un humano. Sin embargo el estudio utiliza una serie de restricciones rítmicas y armónicas en la extracción y generación del material sonoro que merecen ser ampliadas y abordadas de una forma más general. Schulze y Van Der Merwe (2010) utiliza lexicón formado por el cifrado, en número romanos, de un reducido número de acordes; no tiene en cuenta la presencia de silencios; utiliza figuras rítmicas que excluyen tanto tresillos como cualquier otra figuración rítmica irregular y por último limita las figuras rítmicas que continúan de un compás a otro. Estas limitaciones restan complejidad e interés musical al material musical generado computacionalmente por el sistema.

[1] Los investigadores constatan que un orden demasiado elevado produce *overfitting* o sobreentrenamiento en los resultados obtenidos.

Buys y Merwe (2012) proponen un método para la armonización de corales a la manera de J. S. Bach, imitando su estilo mediante la modelización probabilística de la armonía procedente del análisis de distintos corales y la realización de un modelo de transductores de estados finitos. Para la clasificación de los acordes y la extracción de características desde el corpus de entrenamiento se utilizan únicamente doce acordes mayores y menores y se asume además que existe un acorde distinto en cada tiempo del compás. Estas restricciones excluyen acordes complejos, como los acordes de séptima y deja a un lado las posibles inversiones que los acordes triadas o cuatríadas pueden tener, resultando en un reduccionismo demasiado simple para la caracterización de un estilo armónico en particular. Tampoco afronta la vertiente rítmica del estilo musical, creando melodías únicamente mediante la sucesión de la unidad rítmica de la negra.

Roig y col. (2014) proponen un sistema capaz de generar melodías desde un corpus de entrenamiento. Las características que el sistema es capaz de extraer incluyen patrones rítmicos y contornos melódicos, obteniendo que menos del 29 % de los encuestados es capaz de distinguir correctamente al menos 2 de 3 canciones generadas por ordenador de las generadas por un humano. Sin embargo las progresiones armónicas utilizadas están restringidas únicamente a las más frecuentes dentro de la música académica.

Zheng y col. (2017) proponen un sistema basado en cadenas de Markov para la generación de música tradicional china, extrayendo características de altura y duración desde un corpus de música tradicional. Generan líneas melódicas restringiendo su aplicación a escalas pentatónicas y únicamente a la generación de material melódico.

Del análisis crítico de los artículos más relevantes encontramos que en el panorama de la investigación en emulación de estilos musicales es necesario diseñar un modelo que sea capaz de tener en cuenta la máxima riqueza rítmica y armónica posible, para que la caracterización de un estilo musical sea más fidedigna. En concreto, los estilos musicales relativos a la música clásica y al jazz contienen numerosas posibilidades armónicas, resumidas a lo largo de muchos tratados de armonía que establecen cuáles de las combinaciones armónicas son más factibles que otras. Esta riqueza requiere de un sistema que sea tanto capaz de extraer información de cualquier acorde en forma de cifrado como de generar las notas (e inversión de las mismas) relativas a ese cifrado. De manera análoga, el ritmo se puede generar mediante muy variadas células y figuraciones, en las cuales el silencio y los valores irregulares han de ser tenidos en cuenta, sin ningún tipo de restricción. Los estudios analizados en el estado de la cuestión muestran carencias o limitaciones en las vertientes armónicas y rítmicas, resultando demasiado simples para la generación de material musical de interés.

En la presente investigación trataremos de incorporar estas mejoras, diseñando un modelo de lenguaje musical basado modelos de Markov armónicos y rítmicos, de carácter más general, capaces de incorporar un gran número de acordes, trabajar con sus distintas inversiones, distinguir notas de silencios, y reconocer cualquier figura rítmica, extrayendo las características de ritmo y armonía de forma independiente, sin ningún tipo de limitación, para posteriormente generar melodías conforme a estructuras formales preestablecidas por el usuario, que conformarán el armazón de la pieza. Incluiremos en nuestro proyecto el diseño e implementación de una ontología pesada para el control de las estructuras formales y armónicas de las obras generadas. Dicha ontología incluye el conjunto de clases y reglas necesarias para describir las principales características formales de los distintos estilos musicales estudiados en la presente investigación.

1.1.3 Motivación

Desde el punto de vista de mi propio perfil como compositor e investigador en Tecnología Musical y Composición musical asistida por ordenador, la emulación de estilos es un objetivo *per se* que nos ofrece un amplio abanico de retos intelectuales y nos proporciona conocimiento base sobre el que fundamentar nuevas investigaciones. Los sistemas basados en modelos de Markov se han mostrado útiles para la generación de material musical tal y como se ha puesto de manifiesto en otras investigaciones que han obtenido resultados prometedores. Pensamos, por tanto, que es un camino exitoso que merece la pena continuar y expandir con nuevos modelos que sean capaces de incorporar, poco a poco, características musicales más complejas capaces de representar la música en su totalidad.

Desde el punto de vista del compositor, la emulación de estilos mediante la computación nos permite pensar en aplicaciones realmente interesantes. La investigación sobre un sistema inteligente capaz de generar música conforme unas determinadas características estilísticas aprendidas de antemano proporciona al compositor del siglo XXI nuevas herramientas para generar música: el compositor podría ofrecer al sistema como corpus de entrenamiento un corpus construido con sus propias obras, encontrando que el sistema sería capaz de escribir música de manera similar a como el propio compositor lo haría; puede también ofrecer al compositor la posibilidad de entrenar *ad hoc* al sistema para que componga bajo unas determinadas características establecidas por los intereses compositivos el propio compositor, de tal manera que es éste el que evalúa los resultados obtenidos de numerosos experimentos y decide qué material musical utilizar. En términos generales, la composición musical asistida por estos sistemas ofrece material musical que quizá no hubiera sido creado por la imaginación del compositor, ampliando de esta manera la creatividad compositiva del ser humano en una especie de hibridación

entre creatividad humana y computacional. Puede que éstas sean, quizá, las justificaciones últimas que nos animan a investigar en este campo para su posterior aplicación a la composición musical de nuevas obras musicales.

1.2 Problema de investigación e hipótesis

El problema de investigación del presente trabajo es el diseño de un modelo de lenguaje musical basado en cadenas de Markov capaz de incorporar las características armónicas y rítmicas de un determinado estilo musical a partir de un corpus de entrenamiento, para posteriormente generar música emulando el estilo con el que el sistema ha sido entrenado.

Nuestras hipótesis de trabajo se podrían resumir en los siguientes puntos, que resumen los elementos inéditos existentes en nuestra investigación:

- Es posible mejorar los modelos de n-gramas para la armonía y el ritmo trabajando con un lexicón que incluya el mayor número posible de símbolos.
- Mediante el modelo del bigrama y el trigrama es posible generar obras emulando un determinado estilo sin caer en el sobreentrenamiento.
- Podemos realizar un modelo de Markov para un estilo musical calculando independientemente los aspectos armónico y rítmico, es decir, armonía y ritmo son musicalmente independientes.
- Es posible controlar la estructura formal de las obras generadas por nuestro modelo de estilo musical mediante el diseño e implementación de una ontología musical que reúna clases musicales, relaciones y restricciones propias de la música funcional.

1.3 Objetivos

En el presente trabajo de investigación relativo al Máster en I. A. Avanzada, se integrarán determinadas tecnologías existentes de forma original, muchas de ellas estudiadas en contenidos de algunas de las asignaturas del mismo. Incorporaremos mejoras en la extracción de características musicales para describir de manera más exacta el modelo de lenguaje musical, eliminando muchas de las restricciones que aparecen en trabajos de investigación previos, y realizando una aportación interesante e inédita en el campo de la composición musical asistida por ordenador.

Los objetivos del presente trabajo de investigación pueden ser resumidos en los siguientes puntos:

1. Utilizar modelos probabilistas para la emulación de estilos musicales, en concreto el modelo de Markov, de manera independiente para la armonía y para el ritmo, incorporando el mayor número de elementos armónicos y rítmicos, sin las restricciones encontradas en anteriores investigaciones.

2. Diseñar una *ontología pesada* en el dominio de la teoría musical, en el cual se refleje una *base de conocimiento* constituida por el conjunto de reglas para la generación de la armonía, el ritmo, la melodía, el acompañamiento y la estructura formal de la música tonal. Aplicar dicha ontología a la regulación de las estructuras armónicas y formales de las producciones obtenidas por los modelos probabilistas.

3. Utilizar las técnicas propias del procesamiento de lenguaje natural al lenguaje musical, tanto en su vertiente armónica como rítmica.

4. Investigar en el campo de las ontologías en música, realizando aportaciones a determinados trabajos previos existentes[2].

5. Utilizar ontologías estandarizadas para la representación simbólica de la música, utilizándolas tanto en la entrada como en la salida de material musical y enmarcando nuestra propuesta en el contexto de la web semántica.

6. Proporcionar al compositor nuevas herramientas para la creación musical asistida por ordenador.

[2]Sirvan de ejemplo los artículos An ontology for abstract, hierarchical music representation de Harley y Wiggins (2015), Formalizing Quality Rules on Music Notation–an Ontology-based Approach de Cherfi y col. (2017), o A music theory ontology de Rashid, De Roure y McGuinness (2018).

1.4 Metodología de investigación

A continuación expondremos la metodología en la que se enmarcan las distintas fases de la investigación desarrollada en el presente trabajo de fin de máster.

1.4.1 Diseño de la Ontología Musical

El diseño de esta ontología musical pesada y su posterior implementación en el proceso de generación de producciones musicales constituye una de las principales aportaciones del presente trabajo de investigación. Para el desarrollo de una Ontología musical que contenga las clases relativas a la estructura formal y armónica de la composición musical, así como sus principales reglas se han realizado las siguientes fases:

- Adopción de la metodología *Methontology* (Fernández-López, Gómez-Pérez y Juristo, 1997) orientada para la creación de ontologías.
- Elicitación del conocimiento experto. En este caso el conocimiento experto proviene de mi propio conocimiento como Titulado Superior en Música, especialidad en Composición, así como de la consulta de numerosas fuentes bibliográficas.
- Estudio de bibliografía clásica relativa a la armonía musical (Piston, 1989), (Zamacois, 1979).
- Estudio de la relación existente entre melodía y armonía mediante bibliografía de referencia (Toch, 1989), (Schonberg, 1987).
- Análisis de los elementos clave en la constitución del discurso musical como tonalidad, carácter, armonía, ritmo y melodía, así como su jerarquía e interrelación durante el proceso de la composición musical.
- Identificación de los términos necesarios, así como las relaciones y reglas existentes entre ellos, para conceptualizar la ontología pesada.
- Implementación de la ontología en el lenguaje *OWL*, utilizando el software *Protégé*.

1.4.2 Modelado del lenguaje musical

Para diseñar un modelo de lenguaje musical de forma análoga a como lo haríamos en el caso de un lenguaje natural hemos de seleccionar las características musicales que consideremos definitorias de un estilo musical. Nuestra aportación consistirá en analizar de forma independiente las características armónicas y rítmicas, así como su interrelación, incorporando el máximo número posible de símbolos en los diccionarios armónicos y rítmicos de tal manera que pueda verse reflejada la riqueza de los lenguajes musicales. Los pasos desarrollados para esta labor han sido los siguientes:

- Selección de características armónicas: cifrado romano independiente de la tonalidad.
- Selección de características rítmicas: valor de duración en número de negras y etiquetado para distinguir notas de silencios.
- Diseño del modelo de lenguaje musical a partir de modelos de Markov independientes para los planos armónico y rítmico.
- Selección de un estándar de notación para la información musical que contenga de forma representativa estos elementos musicales. Seleccionamos el formato *MusicXML* ya que es un formato abierto de notación musical basado en XML, construido mediante *definiciones de tipos de documento*[3] (DTD) que pueden ser distribuidos libremente bajo la licencia *MusicXML Document Type Definition Public License* (Good, 2001a).
- Análisis de las necesidades de extracción de características musicales. Búsqueda de un *framework* para la extracción de información musical a partir de los archivos MusicXML.
- Minería de corpus: Entrenamiento del modelo musical para cada estilo. Búsqueda y creación de distintos *corpus* musicales en formato MusicXML en los que se encuentren representados distintos estilos musicales y que contengan un gran número de piezas. Selección de las obras incluidas en los corpus de entrenamiento y de evaluación.
- Implementación del módulo de extracción de características musicales armónicas y rítmicas desde el corpus de training mediante *music21 Toolkit*.

[3]Las DTD han sido contenido de estudio de algunas asignaturas del presente máster, así como el toolkit NLTK.

- Implementación del módulo de entrenamiento de los modelos de lenguaje armónico y rítmico mediante el paquete *Natural Language Toolkit* (v.3.4.1) utilizado en distintas asignaturas del máster.

1.4.3 Desarrollo del sistema compositivo

El desarrollo del sistema compositivo constituye la parte central del núcleo de la presente investigación. Consiste en la creación de un sistema de capaz de generar melodías y su acompañamiento armónico mediante un modelo de lenguaje musical markoviano. Para esto se ha llevado a cabo la siguiente metodología:

- Estudio de los diferentes frameworks existentes para el análisis musical asistido por ordenador y el *Music Information Retrieval* (Downie, 2019), (Tjoa, 2019), (Cuthbert y col., 2019).
- Implementación en python del módulo de generación de material musical.
- Implementación de las definiciones y restricciones establecidas en la ontología musical para el control de la forma musical y las estructuras armónicas generadas.
- Implementación del módulo de exportación del material musical a formato estandarizado MusicXML.

1.4.4 Evaluación de resultados

Para la evaluación de los resultados compositivos generados por nuestro sistema se realizarán tres encuestas a un grupo de evaluadores, sobre un conjunto de obras de evaluación, formadas por obras de procedencia humana y de procedencia computacional.

El grupo de evaluadores estará formado por alumnos de Conservatorio Superior de Música, seleccionados por conveniencia de una manera por tanto no aleatoria. Dicha selección implica un alto grado de conocimiento de dominio por parte de los evaluadores, ya que se trata de estudiantes muy bien entrenados en música.

La primera encuesta tendrá el objetivo de determinar la identificación del estilo. La segunda encuesta será un test de Turing para evaluar la capacidad de los evaluadores de distinguir la procedencia de una determinada composición. La última encuesta será un test de preferencia donde los evaluadores tendrán que elegir la obra que prefieren de un conjunto de tres obras (dos computacionales y una

humana), sin conocer la procedencia compositiva de las mismas. Los resultados de estas encuestas serán comparados con encuestas similares realizadas en otros estudios del mismo ámbito investigador, para evaluar de esta manera la capacidad emuladora de nuestro sistema.

1.4.5 Repositorio y control de versiones

Para la correcta gestión del desarrollo del proyecto se ha utilizado una política de gestión del control de versiones, que nos ha permitido controlar de manera organizada los cambios realizados en el código. Durante la fase de desarrollo, para el control de versiones del código programado en python se ha utilizado *Apache Subversion (SVN)* conectado a un repositorio local, donde podemos ver la lista de los diferentes *commits* con una descripción detallada de los cambios realizados. Subversion permite también la comparación de archivos de diferentes versiones, para su cotejo. Una vez el proyecto ha sido finalizado, se ha subido a un repositorio público de GitHub, bajo licencia de software libre GNU GLPv3, a través de la siguiente url:

`https://github.com/BrianComposer/HarmonicVoice`

1.4.6 Uso de Software Libre

Para el desarrollo de nuestro sistema hemos utilizado únicamente soluciones de software libre, contribuyendo a incrementar el uso de estas herramientas y fomentando la distribución de nuestro trabajo, ya que permitimos que cualquier investigador interesado en nuestra aportación pueda tener acceso al código que hemos desarrollado. En la siguiente tabla podemos ver un resumen de las herramientas utilizadas, así como de la licencia que cada una de ellas posee.

Tabla 1.1: Resumen del software libre utilizado en el proyecto.

Software	Licencia
python 3.7	Python Software Foundation License (PSFL)
PyScripter IDE	MIT License
music21 Toolkit	Lesser GNU Public License (LGPL)
music21 Corpus	BSD License
NLTK	Apache License 2.0
Apache Subversion (SVN)	Apache/BSD
MuseScore	GNU GPLv2
MusicXML 3.0	W3C Community Final Specification Agreement
Protégé 5.5	BSD 2-clause

1.5 Estructura del documento

La estructura de la presente memoria de investigación es la siguiente:

- **Capítulo 1 - Introducción:** En este capítulo realizaremos una contextualización a nuestra investigación, exponiendo el problema de investigación, las hipótesis de trabajo, los objetivos y metodología empleada.
- **Capítulo 2 - Estado de la cuestión:** En este capítulo realizaremos una revisión crítica de la literatura científica existente relacionada con la generación de material musical utilizando de modelos de Markov, haciendo especial énfasis en aquellos destinados a la emulación de estilos musicales.
- **Capítulo 3 - Marco teórico:** En este capítulo explicaremos los fundamentos teóricos necesarios para nuestra investigación, así como las técnicas procedentes de la Inteligencia Artificial que serán utilizadas.
- **Capítulo 4 - Tecnologías del proyecto:** En este capítulo explicaremos cada una de las tecnologías utilizadas en nuestra investigación.
- **Capítulo 5 - Diseño de una Ontología musical:** En este capítulo explicaremos el diseño de una ontología pesada construida sobre el dominio de la composición musical, en la que se encuentren definidos los conceptos estructurales necesarios para describir la forma musical, así como las reglas armónicas existentes para la coherencia del discurso musical.
- **Capítulo 6 - Desarrollo del proyecto:** En este capítulo explicaremos el desarrollo de nuestro sistema, explicando la arquitectura del proyecto, la creación de los distintos corpus musicales, la creación del modelo de lenguaje musical y la implementación del sistema compositivo.
- **Capítulo 7 - Resultados:** En este capítulo mostraremos algunos ejemplos de composiciones obtenidas con nuestro sistema, para a continuación exponer los resultados de la evaluación del mismo mediante la realización de varias encuestas, comparando los resultados obtenidos con otros estudios.
- **Capítulo 8 - Conclusiones:** En este capítulo expondremos de forma crítica las principales conclusiones de la presente investigación, comparando los resultados obtenidos con resultados de investigaciones similares. Expondremos las limitaciones aparecidas durante el proceso de investigación, así como futuras líneas de investigación.
- **Referencias bibliográficas**

Capítulo 2

Estado de la cuestión

A continuación expondremos de forma cronológica los resultados de nuestro trabajo de revisión bibliográfica sobre la composición musical asistida por ordenador, poniendo de manifiesto las técnicas de Inteligencia Artificial utilizadas para ello y haciendo especial énfasis en las investigaciones más relevantes que han utilizado modelos de Markov para la generación de nueva música mediante el uso de las computadoras.

2.1 Investigaciones pioneras

En 1955, los autores Caplin y Prinz proporcionan, con anterioridad a los trabajos de Hiller e Isaacson y de Klein y Bolitho, los primeros experimentos documentados de composición asistida por ordenador. Prinz realiza experimentos durante finales de la década de los cincuenta utilizando el computador *Ferranti Mercury*, culminando en la composición *The Foggy, Foggy Dew* (Ariza, 2011). Para la composición de la melodía de esta obra se utilizaron tablas de probabilidad de transición, de forma análoga a una cadena de *Markov* de segundo orden (Hiller, 1968).

Casi simultáneamente, los autores Brooks Jr y col. (1957) proponen un método de composición de melodías basado en las cadenas de Markov, analizando un total de 37 melodías. Varían el orden de la cadena desde orden dos y tres hasta orden ocho, utilizando las computadoras para realizar los necesarios conteos. Incorporan, ade-

más, distintas restricciones para configurar aquellos ritmos melódicos requeridos. Generan un total de 600 melodías, observando el siguiente resultado: las melodías generadas utilizando órdenes muy bajos presentan un alto grado de aleatoriedad, mientras que aquellas que son generadas con probabilidades de órdenes muy altos producen melodías equivalentes a conectar fragmentos de las melodías originales.

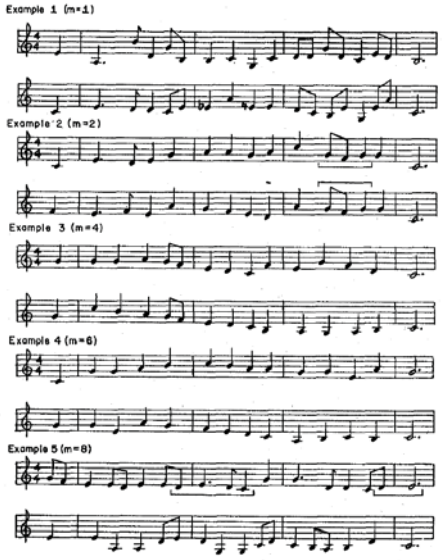


Figura 2.1: Ejemplo de algunas de las melodías obtenidas para distintos órdenes de Markov (Brooks Jr y col., 1957, pág. 181).

Los resultados de los órdenes intermedios presentan resultados con una mayor originalidad aunque producen, en ocasiones, melodías de poco interés musical. Es necesario por tanto un filtrado humano para discriminar las piezas interesantes de aquellas que no lo son. Este modelo basado en cadenas de Markov no es todavía lo suficientemente complejo como para producir música compatible con los requerimientos estéticos deseables.

Entre estos primeros experimentos de la década de los cincuenta destaca el trabajo del compositor Lejaren Hiller en la Universidad de Illinois, junto con su colaborador Leonard Isaacson, matemático empleado de la compañía *Standard Oil Company*. Hiller trabaja con el computador *ILLIAC I (Illinois Automatic Computer*, construido en 1952 y consistente en varias máquinas que sumaban un total de cinco toneladas de peso y aproximadamente 2,800 válvulas de vacío, programable mediante cinta de papel perforada (Holmes, 2012).

Figura 2.2: Computadora ILLIAC I de la Universidad de Illinois utilizada por Hiller (Hiller e Isaacson, 1959, pág. 109).

Su resultado cristaliza en la *Suite Illiac*, para cuarteto de cuerda, finalizada en noviembre de 1956, considerada como la primera gran composición musical creada con la ayuda de un ordenador. Cada uno de los cuatro movimientos se corresponde con cada uno de los cuatro objetivos fundamentales de la investigación. En este proyecto, Hiller explora la utilización de métodos probabilistas basados en modelos de Markov (Hiller e Isaacson, 1959; Hiller, 1959).

En 1975 el investigador Tipei presenta el programa *MP1*, orientado a la composición musical asistida por ordenador, diseñado como un asistente capaz de ayudar al compositor en sus labores cotidianas (Tipei, 1975). Trabaja bajo el supuesto de que cualquier sucesión continua de sonidos puede ser descrita por una cadena de Markov, ofreciendo la posibilidad de componer música tradicional tonal, música serial y música estocástica.

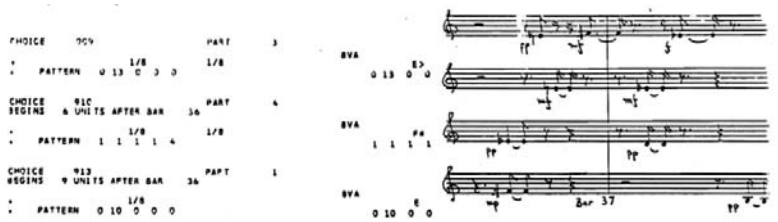


Figura 2.3: Ejemplo de output de MP1 y su correspondiente notación musical (Tipei, 1975, pág. 79).

2.2 Utilización de redes neuronales

Sobre la utilización de redes neuronales en la composición musical asistida por ordenador es destacable el sistema *HARMONET*, creado por Hild, Feulner y Menzel (1992). El sistema permite la armonización de melodías emulando el estilo de los corales de J. S. Bach, utilizando redes neuronales, supeditadas a un conjunto jerárquico de reglas. El sistema se entrena mediante un corpus de doce corales de Bach. Se obtiene la estructura armónica mediante la reducción de cada coral a sus armonías básicas, considerando para ello las corcheas y semicorcheas como notas de adorno, externas a la armonía.

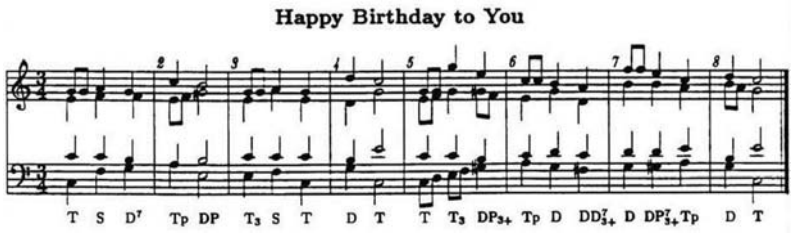


Figura 2.4: *Cumpleaños feliz* armonizado (Hild, Feulner y Menzel, 1992, pág. 273).

En similar línea de investigación, Mozer (1994) presenta en su artículo *Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing* la red neuronal *CONCERT*. El sistema requiere ser entrenado mediante un corpus de melodías de las que es capaz de extraer regularidades como progresiones melódicas y armónicas. Incorpora, además, restricciones de tipo psicológico procedentes de estudios sobre la percepción humana, mostrando resultados que se parecen ser superiores a los resultados obtenidos por modelos de Markov de orden tres.

Otras investigaciones relevantes en este ámbito son las propuestas de Browne y Fox (2009) en la que se generan melodías con una red neuronal artificial para simular los procesos de tensión y relajación, o la propuesta de Coca, Romero y Zhao (2011) en la que se generan melodías mediante un modelo de red neuronal recurrente que utiliza sistemas lineales caóticos para introducir variaciones.

2.3 Utilización de gramáticas

La utilización de gramáticas aplicadas a la composición musical asistida por ordenador ha sido ampliamente estudiada, tal y como podemos constatar en la bibliografía existente al respecto.

Algunos ejemplos son la propuesta de Quick (2010) en la que se realiza contrapunto clásico a tres voces integrado en un framework Shenkeriano[1]. Otra relevante aportación es la propuesta por Keller y Morrison (2007) en la que se diseña un sistema capaz de realizar improvisaciones en estilo jazz mendiante el uso de gramáticas formales y la implementación de una potente interfaz de usuario.

2.4 Utilización de algoritmos genéticos

Las técnicas procedentes de la computación evolutiva también han sido profusamente utilizadas para la composición musical asistida por ordenador. A continuación expondremos algunos ejemplos.

Los autores Falkenstein y Tlalim (2009) presentan el sistema *Alter Ego* programado sobre *SuperCollider*, basado en la implementación de algoritmos genéticos junto con modelos de Markov. El sistema es capaz de codificar parámetros como la amplitud, el tiempo de ataque, el tiempo de *release*, las curvas de desvanecimiento, o el panorámico, entre muchos otros, para la posterior generación de material musical.

Otras investigaciones relevantes en este ámbito son Jensen (2011), en la que se generan melodías utilizando programación genética que incorpora una función de fitness que evalúa la distancia con respecto a un determinado corpus, o la propuesta de Dahlstedt (2007) en la que se genera música clásica contemporánea utilizando métodos genéticos y una compleja función de fitness que combina múltiples características musicales.

[1]Para más información sobre el análisis Shenkeriano véase `https://en.wikipedia.org/wiki/Schenkerian_analysis`.

2.5 Utilización de modelos de Markov

A continuación nos centraremos en la utilización de modelos de Markov y métodos similares para la composición musical asistida por ordenador. El programa *Melody-one*, expuesto por Abel (1981) en su artículo *Computer Composition of Melodic Deep Structures*, realiza la generación de melodías mediante una hibridación entre cadenas de Markov y gramáticas formales. Los resultados son melodías agradables que encajan con los criterios de la armonía tonal occidental. El procedimiento para la generación de las melodías es una cadena de Markov controlada por una gramática, que establece si la nota generada está permitida o no.

Jones (1980) muestra la utilización de estructuras motívicas como estados posibles de un modelo de Markov, en lugar de notas musicales (Nierhaus, 2009).

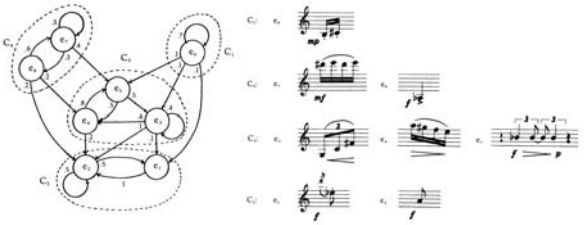


Figura 2.5: Modelo de Markov (izquierda) y motivos como elementos terminales (derecha) (Jones, 1980, pág. 49).

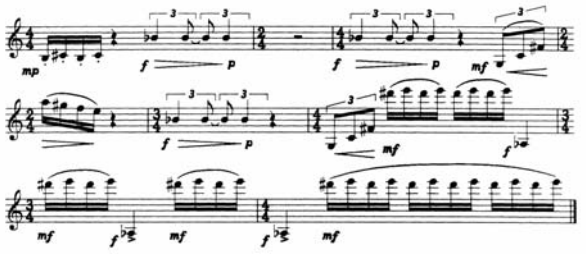


Figura 2.6: Ejemplo de producción por el sistema propuesto por Jones (Jones, 1980, pág. 50).

El autor Meinecke (1981) desarrolla un sistema de algoritmos que incluyen la generación de alturas y de ritmos además de utilidades para el análisis musical asistido por ordenador. Para la determinación del intervalo musical se utiliza un proceso estocástico de segundo orden basado en una cadena de Markov tridimensional. El algoritmo selecciona la dirección del intervalo en función de una serie de cuatro contornos melódicos predefinidos. Obtiene, como resultado destacable, que

el estilo de las melodías generadas depende de las melodías que ha sido utilizadas en el *corpus de entrenamiento*.

Los autores Cruz-Alcázar y Vidal-Ruiz (1998) proponen un sistema de generación melódico en el cual la extracción de características musicales se realiza mediante gramáticas formales, y la producción musical se efectúa mediante modelos de Markov. Los autores aplican su método sobre un total de tres estilos musicales diferentes, para generar melodías nuevas, de manera estocástica.

Martin Hirzel y Daniela Soukup (2000) generan un sistema capaz de realizar improvisaciones de jazz basadas en pequeños patrones que son procesados por un Modelo Oculto de Markov (HMM). El usuario debe introducir los patrones que forman el repertorio del sistema y éste los transpondrá para adaptarlos a las progresiones armónicas generadas (Nierhaus, 2009).

Los autores Triviño-Rodriguez y Morales-Bueno (2001) proponen un sistema de generación automatizada de melodías basado en un modelo avanzado de Markov denominado *Probabilistic Suffix Automata* (PSA). El sistema consiste en cadenas markovianas de longitud variable. Las melodías generadas a partir de un corpus de entrenamiento, compuesto por cien corales de J. S. Bach, muestran una alta similitud con las melodías compuestas por Bach.

Los autores Farbood y Schöner (2001) presentan un novedoso método para la generación automática de contrapunto, emulando el estilo de Palestrina, a partir de una melodía o *cantus firmus* proporcionado por el usuario. El sistema está basado en cadenas de Markov de orden dos. Cada una de las estrictas reglas compositivas existentes en la disciplina del contrapunto se ve implementada en el sistema como una tabla de probabilidad distinta, donde las transiciones interválicas que no están permitidas tienen una probabilidad igual a cero, existiendo por tanto la *tabla armónica*, la *tabla melódica*, la *tabla cadencial*, la *tabla cromática*, la *tabla de movimientos paralelos*, la *tabla de movimientos cercanos*, la *tabla de movimiento de salida*, la *tabla de movimiento general*, y finalmente *tabla de clímax*.

	m2u	M2u	m3u	M3u	P4u	P5u	m6U	P8u	m2d	M2d	m3d	M3d	P4d	P5d	P8d	P1
m2u	0	.45	.2	.2	0	0	0	0	.035	.035	.025	.025	.01	.01	.009	.001
M2u	.45	.45	.03	.03	0	0	0	0	.01	.01	.005	.005	.004	.004	.002	.001
m3u	.45	.45	.03	.03	0	0	0	0	.01	.01	.005	.005	.004	.004	.002	.001
M3u	.35	.35	.05	.05	0	0	0	0	.05	.05	.025	.025	.025	.025	.024	.001
P4u	.065	.065	0	0	0	0	0	0	.4	.4	.02	.02	.01	.01	.009	.001
P5u	0	0	0	0	0	0	0	0	.52	.52	.01	.01	.01	.01	.009	.001
m6u	0	0	0	0	0	0	0	0	.51	.51	.025	.025	.01	.01	.009	.001
P8u	0	0	0	0	0	0	0	0	.51	.51	.025	.025	.01	.01	.009	.001
m2d	.05	.05	.005	.005	.002	.002	.002	.002	0	.467	.2	.2	.015	0	0	.001
M2d	.05	.05	.005	.005	.002	.002	.002	.002	.367	.367	.1	.1	.015	0	0	.001
m3d	.366	.366	.005	.005	.002	.002	.002	.002	.05	.05	.1	.1	0	0	0	.001
M3d	.366	.366	.005	.005	.002	.002	.002	.002	.05	.05	.1	.1	0	0	0	.001
P4d	.53	.53	.02	.02	.02	.02	.039	.02	0	0	0	0	0	0	0	.001
P5d	.454	.454	.03	.03	.01	.005	.005	.011	0	0	0	0	0	0	0	.001
P8d	.454	.454	.03	.03	.01	.005	.005	.011	0	0	0	0	0	0	0	.001
P1	.1	.1	.08	.08	.05	.05	.04	.03	.1	.1	.08	.08	.05	.04	.02	0

Figura 2.7: Tabla para las reglas de transición melódica (Farbood y Schöner, 2001, pág. 3).

Moray Allan (2002) propone un sistema basado en modelos de Markov y modelos ocultos de Markov para la armonización de melodías. Los cospus de entrenamiento y test se generan mediante el conjunto de los corales de J. S. Bach (Nierhaus, 2009).

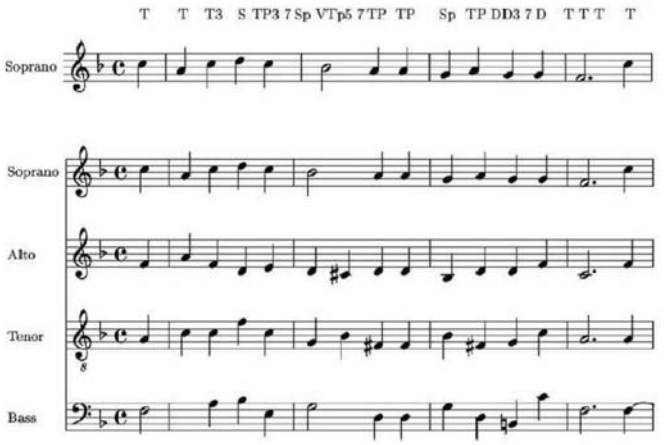


Figura 2.8: Armonización del coral *Dank sei Gott in der Höhe*, BWV 288 de J. S. Bach realizada por el sistema de Allan (Allan, 2002, pág. 49).

Schulze y Van Der Merwe (2010) proponen un sistema en el que se realiza la emulación de estilos mediante el cálculo de n-gramas de distintos órdenes (hasta orden diez) sobre un corpus de entrenamiento, para realizar a posteriori nuevas composiciones. Los resultados obtenidos resultan prometedores, encontrando que un 39 % los encuestados es incapaz de distinguir composiciones generadas por el ordenador con las generadas por un humano.

Sertan y Chordia (1975) utilizan el modelo de Markov de longitud variable que considera el tono, el ritmo, instrumento, y clave, para predecir la secuencia posterior de la música popular turca. Por otra parte A. Prechtl y Samuels (2014) generan música para videojuegos mediante el uso de las cadenas de Markov con características musicales tales como tempo, velocidad, volumen y acordes (Liu y Ting, 2017).

En el artículo *Generating and evaluating musical harmonizations that emulate style* los autores Chuan y Chew (2011) exponen un sistema híbrido para la emulación de estilos aplicada a la armonía. Se combinan las cadenas de Markov con un conjunto de reglas teóricas y musicales, siendo el sistema capaz de inferir estas

reglas y generar el modelo markoviano desde el corpus de entrenamiento, a partir de líneas melódicas en formato MIDI, con la armonía expresada como acordes etiquetados en formato de texto. Mediante árboles de decisión el sistema es capaz de aprender las relaciones melódico-armónicas de un determinado estilo musical.

En su artículo Feature Extraction and Machine Learning on Symbolic Music using the music21 Toolkit, los autores Cuthbert, Ariza y Friedland (2011) explican cómo el framework *music21* puede utilizarse en técnicas propias del *machine learning* aplicadas tanto a la composición como al análisis musical asistido por ordenador. Este toolkit ofrece una serie de sofisticadas funcionalidades para la extracción de características musicales.

Los autores Manaris y Hughes (2011) proponen *Monterrey Mirror*, un sistema para generación de música estocástica basado en modelos de Markov junto con algoritmos genéticos y reglas de composición, combinando el poder predictivo de los modelos markovianos con la capacidad de generación de nuevo material que proporcionan las técnicas de la computación evolutiva.

Pachet, Roy y Barbieri (2011) proponen un innovador sistema de generación controlada de melodías mediante modelos de Markov que integra restricciones en el mismo. Los autores comparan melodías de jazz creadas por un compositor humano con melodías generadas mediante el sistema, en las que determinadas notas son fijadas para que coincidan.

Los autores Cherla, Purwins y Marchini (2013) proponen un software capaz de generar variaciones a partir de una melodía proporcionada. Se utilizan técnicas de aprendizaje no supervisado, cadenas y modelos ocultos de Markov. Los resultados obtenidos, sometidos a revisión por un panel de expertos mediante un cuestionario de evaluación, obtienen opiniones favorables.

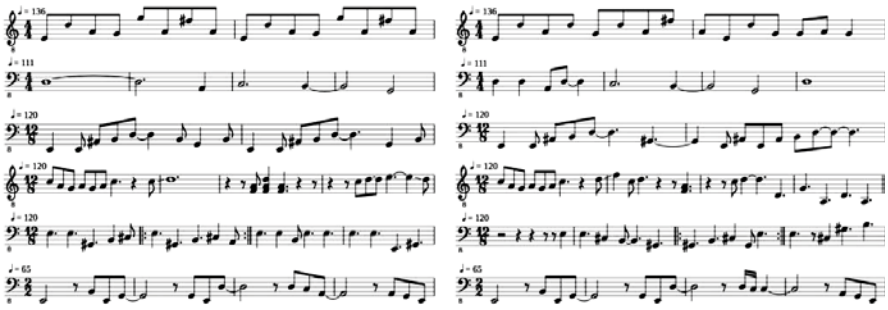


Figura 2.9: Comparación entre melodías iniciales (derecha) y sus variaciones (izquierda) (Cherla, Purwins y Marchini, 2013, pág. 76).

McVicar, Fukayama y Goto (2014) presentan el software *AutoRhythmGuitar*, orientado a la composición asistida por ordenador. El programa genera ritmos de guitarra con sus correspondientes tablaturas, tomando como parámetro de entrada una secuencia armónica. El sistema está basado en modelos de Markov y se entrena un conjunto de tablaturas procedente de cinco guitarristas famosos. Los resultados generados se exportan como partitura convencional en formato MusicXML.

Roig y col. (2014) proponen un sistema capaz de generar melodías aprendidas desde un corpus de entrenamiento. Las características que el sistema es capaz de extraer incluyen patrones rítmicos y contornos melódicos. El compositor puede elegir el número de compases de la composición, el compás, el tempo, tonalidad y base de datos de la obra generada. Entrenan el sistema con los estilos pop, académico y flamenco, obteniendo que menos del 29 % de los encuestados es capaz de distinguir correctamente al menos 2 de 3 canciones generadas por ordenador de las generadas por un humano.

En Zheng y col. (2017) se expone un sistema basado en cadenas de Markov para la generación de música tradicional china, extrayendo características de altura y duración desde un corpus de música tradicional y generando líneas melódicas con interesantes resultados, sin embargo se restringe su aplicación a escalas pentatónicas y únicamente a la generación de material melódico.

En Zhu y col. (2018) presentan el sistema *XiaoIce Band* mediante el cual se permite la generación automatizada de canciones multipista en estilo pop, pretendiendo solucionar el problema de la direccionalidad de la secuencia armónica generada y la calidad de los ritmos obtenidos. El sistema muestra un significativo avance en el test de Turing con respecto a otros sistemas similares.

Tabla 2.1: Tabla resumen con las investigaciones más relevantes

Referencia	Tarea compositiva	Comentarios
(Tipei, 1975)	melodía	Utiliza cadenas de Markov como parte de un sistema ad hoc más grande
(Jones, 1980)	Fragmentos musicales	Sistema muy sencillo, orientado a compositores Fragmentos musicales completos
(Abel, 1981)	melodía	Cadenas de Markov junto con gramáticas formales. Melodías muy sencillas
(Cruz-Alcázar y Vidal-Ruiz, 1998)	melodías	Gramáticas formales y modelos de Markov
(Schulze y Van Der Merwe, 2010)	melodía	modelo de n-gramas de distintos órdenes para emular estilos
(Manaris y Hughes, 2011)	Improvisación melódica interactiva	El modelos de Markov genera un candidato para el algoritmo evolutivo.
Pachet, Roy y Barbieri	melodía	integra modelos de Markov y restricciones
(Cherla, Purwins y Marchini, 2013)	generación de variaciones melódicas	Utilización de modelos ocultos de Markov
(McVicar, Fukayama y Goto, 2014)	generación de tablaturas de guitarra	Utilización de n-gramas utilizando corpus de cinco guitarristas famosos
(Roig y col., 2014)	melodías sencillas	extrae contornos melódicos y patrones rítmicos desde un corpus de entrenamiento para calcular los n-gramas
(Zheng y col., 2017)	melodías	Generación de música folklórica china mediante cadenas de Markov
(Zhu y col., 2018)	melodías pop	Generación de canciones multipista en estilo pop, proporcionando direccionalidad armónica

2.6 Conclusiones

Una cadena de Markov es, conceptualmente, una idea simple: es un proceso estocástico que transita en pasos de tiempo discretos a través de un conjunto de estados finitos, sin memoria. El siguiente estado depende únicamente del estado actual, no de la secuencia de estados precedentes, pudiéndose representar en una matriz las probabilidades de transición entre estados. Cuando aplicamos cadenas de Markov a la composición musical asistida por ordenador, las matrices de probabilidad pueden ser deducidas a partir de un corpus de entrenamiento, formado por composiciones preexistentes, o derivadas desde la teoría musical a base de prueba y error. El primer caso es más común en el ámbito de la investigación, mientras que el segundo caso tiene una aplicación más frecuente como herramienta para los compositores.

Es bastante común extender el número de estados anteriores que influyen en el actual, denominando n al orden de esta cadena de Markov. También se puede denominar n-grama, aunque estrictamente hablando nos refiramos a una secuencia de n estados. Como consecuencia, la matriz de probabilidad tiene $n + 1$ dimensiones. En la composición algorítmica, las cadenas de Markov se han usado principalmente como sistemas generativos, aunque también pueden utilizarse como herramientas de análisis.

Las cadenas de Markov han sido muy populares en la composición musical asistida por ordenador desde los inicios de la disciplina, aunque sus limitaciones rápidamente se ponen de manifiesto: cadenas de Markov de órdenes bajos producen composiciones extrañas y poco musicales, que carecen de direccionalidad, mientras que las cadenas de órdenes muy altos producen tienden a reproducir literalmente fragmentos musicales del corpus de entrenamiento y son, además, computacionalmente muy costosas de calcular. Por este motivo, las cadenas de Markov se han utilizado más como generación de material musical sin pulir que como un método para la composición musical automatizada. Las cadenas de Markov se han utilizado en los últimos años hibridadas con otros métodos como por ejemplo el uso de gramáticas formales o algoritmos evolutivos.

En nuestra investigación abordaremos el problema de la composición musical de melodías acompañadas, emulando un estilo a partir de un corpus de entrenamiento de una forma novedosa: se entrenará el modelo generando bigramas y trigramas de manera independiente para la armonía y para el ritmo. Por una parte, gracias a la utilización del novedoso toolkit *music21* para la musicología computerizada[2], incluiremos un sistema de extracción de características armónicas capaz de

[2]Para más información véase `https://web.mit.edu/music21/`

tener en cuenta las inversiones de los acordes, tanto triadas como cuatríadas y quintíadas, mejorando de esta manera muchas de las limitaciones de investigaciones anteriores que reducían el número acordes posibles y permitiendo por tanto incorporar breves modulaciones o *flexiones* a tonos vecinos[3], enriqueciendo con una mayor complejidad las secuencias armónicas generadas. Por otra parte, en el plano rítmico no existirán limitaciones en cuanto a las figuraciones que el sistema es capaz de analizar, siendo capaz de analizar cualquier elemento rítmico de cualquier duración, ya sea nota o silencio. La construcción de la melodía acompañada se realizará mediante la superposición de una secuencia rítmica sobre la secuencia armónica previamente generada, estableciendo un sistema de reglas para la selección de las notas compatibles con cada acorde. Se diseñará una *Ontología musical pesada*, elaborada previamente mediante elicitación de conocimiento experto, que formularán las definiciones de clases y reglas entre las mismas necesarias para la regulación de las estructuras armónicas y formales de las producciones musicales obtenidas por el sistema. Las composiciones por tanto se agruparán en frases y semifrases, que comienzan con determinados acordes permitidos y finalizan con una determinada cadencia permitida, preestablecidas por las reglas ontológicas, evitando de esta manera el comportamiento errático de las composiciones musicales generadas con órdenes de Markov bajos. Las definiciones de clases, reglas y restricciones formuladas a modo de ontología pesada, contribuirán además a la investigación en ontologías musicales. Nuestra propuesta pretende obtener composiciones completas y coherentes desde el punto de vista formal y armónico, emulando al mismo tiempo un determinado estilo musical o compositor.

[3]Por ejemplo acordes dominantes secundarias, armonías alteradas, etc.

Capítulo 3

Marco teórico

En este capítulo explicaremos los conceptos y técnicas que han sido necesarias para el desarrollo de la investigación. Para la generación de material musical se realizará previamente un entrenamiento de un modelo armónico y rítmico, gracias a la extracción de características musicales desde el análisis de distintos corpus de entrenamiento. Utilizaremos en el ámbito musical muchas de las técnicas propias de la disciplina de procesamiento del lenguaje natural descritas en esta sección, como la *tokenización* o la segmentación, así como las técnicas de conteo y suavizado de n-gramas. En nuestra investigación utilizaremos determinados aspectos de la morfología lingüística aplicados al lenguaje musical para poder seleccionar y extraer las características musicales definitorias de un estilo musical, por tanto realizaremos una introducción detallada a dicha disciplina exponiendo los conceptos y técnicas necesarios para su correcta explicación, pasando por los autómatas finitos, de especial relevancia musical por su conveniencia a la hora de representar posibles encadenamientos armónicos o procesos cadenciales estandarizados dentro de un estilo musical; transductores de estados finitos; o expresiones regulares, empleadas en nuestro trabajo para la búsqueda de ocurrencias de determinados símbolos musicales dentro de una obra. Por último hablaremos sobre los modelos de Markov y los modelos ocultos de Markov, utilizados en profusión para la composición musical asistida por ordenador, exponiendo su relación con los puntos anteriormente tratados.

3.1 Expresiones regulares

Comencemos hablando de las *expresiones regulares* a modo de prólogo para el estudio de los autómatas finitos. Dichas expresiones son una notación estándar para caracterizar secuencias de texto inicialmente desarrolladas por Kleene en 1956. Constituyen secuencias de caracteres determinados que pueden utilizarse como patrón de búsqueda, introduciendo determinadas reglas o restricciones sobre el conjunto total de cadenas de texto que serán filtradas. Podemos decir por tanto que las *expresiones regulares* constituyen un lenguaje para especificar restricciones en la búsqueda de textos con el que se pueden definir patrones en la misma. De esta manera, la búsqueda mediante *expresiones regulares* requerirá tanto la definición de un patrón de búsqueda como el establecimiento de un *corpus* de palabras sobre las que ejecutar dicha búsqueda. La función de búsqueda mediante expresiones regulares buscará recorriendo el conjunto de palabras de este corpus, devolviendo como resultado todos los textos o palabras que sean compatibles con dicho patrón (Kleene, 1956).

La expresión regular más sencilla es una simple cadena de texto (de cualquier longitud, inclusive un único carácter). Por ejemplo la expresión regular */edro/* nos proporcionaría resultados del tipo *Pedro*, *dodecaedro*, *dodecaedros*, *cedro*. Podemos generar búsquedas más complejas mediante la utilización de la sintaxis propia de las expresiones regulares, tal y como veremos a continuación.

Los caracteres incluidos entre corchetes *[ ]* especifican una disyunción entre los mismos. El motor de búsqueda devolverá resultados que contengan uno de estos caracteres; así por ejemplo la expresión /[Pp]edro/ devolverá *Pedro* y *pedro*. Podemos incluir un guión dentro de los corchetes para especificar que los caracteres permitidos son los que están dentro de ese intervalo. Por tanto /[c-p]edro/ devolvería *cedro* y *pedro*, pero no *Pedro*, ya que la búsqueda distingue entre mayúsculas y minúsculas (Martin y Jurafsky, 2009).

3.1.1 Algunos operadores importantes

Mediante el símbolo ^ podemos también especificar que un determinado carácter no parezca, por ejemplo /[^c]edro/ devolvería *pedro* y *Pedro*. El comportamiento de ^ es tal que evita que aparezcan los caracteres que están situados a su derecha, es decir, si ^ es el primer símbolo que aparece tras el corchete de apertura [.

El uso del *símbolo de interrogación* nos permite especificar a la búsqueda que pueda aparecer o no el carácter precedente. Por ejemplo, la expresión /perros?/ nos retornará *perro* y *perros*.

Sin embargo si queremos especificar que el carácter anterior puede aparecer cero, una, o muchas veces en la cadena de texto, deberemos utilizar el asterisco. Por ejemplo la expresión /woo*w/ nos devolverá *wow*, *woow*, *wooow*, *woooow*, etc.

El símbolo + nos indicará que el carácter anterior puede aparecer una o muchas veces en la cadena de texto. Por tanto la expresión /wo+w/ sería análoga a la anterior /woo*w/.

El símbolo '.' es un carácter *comodín*; nos indicará que el carácter que esté en esa posición puede ser cualquiera. De esta forma la expresión /cas./ devolverá *casa*, *casi*, *caso*. Si queremos distinguir la expresión regular . del carácter ".", deberemos marcar '.' como carácter de escape utilizando la barra invertida antes del carácter (i.e.: "colorín colorado este cuento se ha acabado\.").

Los delimitadores (*anchors*) restringen la aparición de una cadena en una posición determinada. Los más conocidos son el acento circunflejo ^ , que especifica que la cadena tiene que aparecer al principio; el símbolo del dólar $, que especifica que la cadena tiene que aparecer al final.

Para establecer disyunciones en las búsquedas, además del anteriormente mencionado [ ], tenemos el operador de disyunción |. Utilizado de manera conjunta con los paréntesis, podemos generar expresiones regulares como: /ingl(és|esa)/ para obtener tanto el resultado *inglés* como *inglesa*.

La expresión regular {} se pertenece a los *contadores*, ya que determina el número exacto de veces que un determinado carácter ha de aparecer. Por tanto /3/ nos devolverá exactamente tres ocurrencias del anterior carácter o expresión.

Hay que tener en cuenta que el uso de los operadores descritos en este apartado se regula por una ordenación o *precedencia*. De mayor a menor precedencia, los operadores se ordenan de esta forma: primero los paréntesis; segundo *, +, ? y {}; tercero delimitadores (^ y $) y secuencias; por último la disyunción |.

Dada la alta casuística y la complejidad de los lenguajes naturales, puede darse el caso que necesitemos realizar combinaciones más complicadas de expresiones regulares para buscar una determina cadena de texto sobre un corpus. En estos casos, aparece un compromiso entre la exactitud de los resultados obtenidos y la cobertura de los mismos. Minimizar el número de falsos positivos (aumento de exactitud) resulta no ser compatible con el hecho de minimizar los falsos negativos (aumentos de cobertura). Por tanto en numerosas ocasiones encontraremos casos en los que se verá como la efectividad de la búsqueda se encuentra comprometida con la generalidad de la misma (Martin y Jurafsky, 2009).

3.2 Autómatas finitos

Un autómata finito (FSA) es un modelo matemático-computacional de una máquina que realiza cómputos en forma automática sobre unos símbolos de entrada para producir una salida. Dicha máquina teórica posee una cinta de entrada para en la que se escribe una cadena y un cabezal lector que es capaz de leer los símbolos de uno en uno y de izquierda a derecha. La máquina tiene un conjunto de reglas que le especifica cómo actuar ante cada símbolo de entrada de la máquina. Un autómata finito puede ser representado mediante un grafo dirigido (diagrama de estados), donde los nodos son los diferentes estados de la máquina y los vértices representan las transiciones entre estados. El estado final se representa con un círculo doble (Martin y Jurafsky, 2009). En el siguiente ejemplo podemos ver el grafo de un autómata finito que acepta el lenguaje $\{a^n cb^m / n > 0, m \geq 0\}$

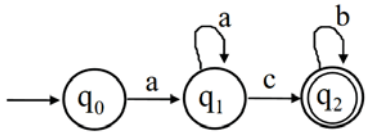


Figura 3.1: Ejemplo de un grafo de un autómata finito (Mauco y Barbuzza, 2009, pág. 1).

Más formalmente, un autómata finito es un conjunto de estos cinco parámetros:

$$\begin{array}{ll} Q = \{q_0, q_1, \ldots, q_{N-1}\} & \text{conjunto finito de N estados} \\ \Sigma & \text{alfabeto finito de símbolos de entrada} \\ q_0 & \text{estado inicial} \\ F & \text{conjunto de estdos finales } F \subseteq Q \\ \delta(q, i) & \text{función de transición entre estados} \end{array} \tag{3.1}$$

La función de transición $\delta(q, i)$ nos devuelve, dados un estado $q \in Q$ y un símbolo $i \in \Sigma$, un nuevo estado $q' \in Q$. Podría ser expresada en forma de *tabla de transiciones*.

La relación entre autómatas finitos y expresiones regulares es muy estrecha: cualquier expresión regular puede representarse como un autómata finito siempre y cuando ésta no utilice el concepto de *memoria*. Al mismo tiempo, cualquier autómata finito puede ser descrito en términos de una expresión regular. Podríamos concluir por tanto que expresiones regulares y autómatas finitos son equivalentes. Además, se puede demostrar que si un lenguaje es descrito por una expresión regular, entonces éste es un tipo particular de lenguaje formal denominado lenguaje regular. Un lenguaje formal es un conjunto infinito de expresiones que pueden ser

generadas a través de un conjunto finito de símbolos llamado *alfabeto*. Por tanto un lenguaje regular será un lenguaje formal que puede ser caracterizado mediante expresiones regulares (Martin y Jurafsky, 2009).

3.2.1 Ejemplos de autómatas finitos

A continuación expondremos a modo de ejemplo los grafos de algunos de los autómatas finitos que pueden deducirse de algunas expresiones regulares en el análisis morfológico:

- La expresión regular *'.*(a|i|e)ré$'*, utilizada para etiquetar la segunda persona del singular del pretérito perfecto simple de algunos verbos (e.g. pasaste, comiste), se puede expresar como un autómata finito de la siguiente manera:

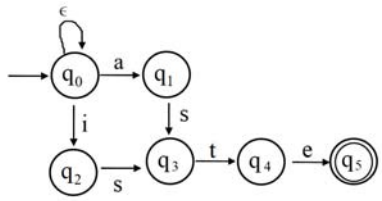


Figura 3.2: Expresión regular como autómata finito. Elaboración propia basada en (Mauco y Barbuzza, 2009, pág. 1).

- La expresión regular *'.*(a|i)ste$'*, utilizada para etiquetar la primera persona del singular del futuro simple de indicativo de algunos verbos (e.g. caeré, callaré, iré), se puede expresar como un autómata finito de la siguiente manera:

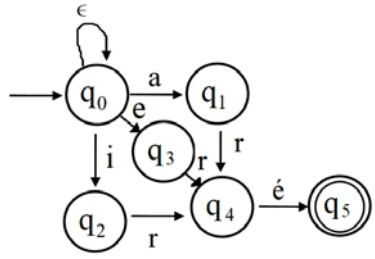


Figura 3.3: Expresión regular como autómata finito. Elaboración propia basada en (Mauco y Barbuzza, 2009, pág. 1).

3.2.2 Autómatas deterministas y no deterministas

Un autómata finito puede tener ramificaciones en el grafo que lo representa. Si en cada bifurcación todos los vértices que abandonan el nodo tienen distintos símbolos, el autómata no tendrá que hacer una decisión para continuar; hablamos en este caso de autómatas finitos deterministas (DFSA), ya que la decisión de qué camino coger viene proporcionada por la lectura del símbolo. En este caso, el comportamiento del autómata vendrá completamente determinado por el estado en el que se encuentra y por el símbolo que está leyendo. Sin embargo, si existen bifurcaciones en las cuales al menos dos vértices comparten símbolo, entonces el autómata tendrá que enfrentarse al problema de decidir cual es el mejor camino para continuar, ya que para un mismo símbolo leído existen al menos dos vértices posibles; éste es el caso de los autómatas finitos no deterministas (NFSA). Otro tipo de autómata no determinista es aquel cuyo grafo contiene vértices en los que no se ha especificado ningún símbolo (transiciones ϵ), dichas transiciones pueden interpretarse como cambios de estado que puede realizar el autómata sin necesidad de leer ningún símbolo (Martin y Jurafsky, 2009).

Contrariamente a lo que se podría suponer, para todo automata finito no determinista existe un equivalente autómata finito determinista. Por tanto se puede demostrar que todo autómata finito no determinista puede ser expresado como un autómata finito determinista, existiendo un algoritmo mediante el cual se puede realizar dicha conversión, aunque el número de estados posibles existentes en el autómata determinista equivalente será mucho más grande. Si queremos comprobar cadenas de texto en el caso de un autómata finito no determinista, se han de introducir mecanismos de búsqueda que realicen las decisiones, ya que cuando se presenta en elección no todos los vértices tienen porqué ser válidos. El orden en el cual un autómata finito no determinista escoge cual es el siguiente estado determina su *estrategia de búsqueda*. Las tres estrategias más extendidas para realizar este proceso son las siguientes (Martin y Jurafsky, 2009):

1. **Backup:** Cada vez que se llega a una decisión se almacena un marcador que nos guarde la información relativa al estado que estábamos. Si al escoger un camino determinado, descubrimos que es el incorrecto, el autómata volverá al anterior punto de decisión y recorrerá otro camino.

2. **Look-ahead:** Antes de seleccionar un determinado camino, se mira unos cuantos estados por delante para determinar si ese camino es conveniente.

3. **Paralelismo:** Cuando llega una decisión, se recorren todos los diferentes caminos en paralelo.

3.3 Morfología lingüística

La morfología es la rama de la lingüística que estudia la estructura de las palabras. Un morfema es una unidad mínima con significado, debiendo tener una palabra cualquiera al menos un morfema (aunque puede tener dos o más). Cada palabra se constituye con un elemento básico de expresa su significado denominado *raíz*. Además una palabra puede contener uno o más *afijos*, elementos añadidos que tienen función gramatical o semántica. Existen muchos tipos de afijos: prefijos (afijos que siguen a la raíz), sufijos (afijos que preceden a la raíz), interfijos (afijos que se introducen en medio de la palabra), infijos (afijos que se introducen en el medio de la raíz) y circunfijos (afijos que se colocan antes y después de la raíz). En inglés se utilizan todos, en castellano muy predominantemente los prefijos y sufijos. Los afijos pueden ser de dos tipos: *derivativos*, que sirven para crear palabras nuevas a partir de la palabra base; o *flexivos*, que cambian la función gramatical de la palabra raíz. Algunos ejemplos de afijos en castellano son los siguientes: *morfemas derivativos:* ante-; post-; extra-, etc. *morfemas flexivos:* -dad, -ento (sufijos sustantivos), -oso, -ante (sufijos adjetivales), -ificar (sufijos verbales), -mente (sufijos adverbiales). *Interfijos:* piece-**ec**-ito. *Infijos:* Carl-**it**-os. No existen *Circunfijos* en la lengua castellana. Existen en castellano una serie de pronombres denominados *enclíticos* que se unen al verbo precedente para formar una única palabra. Un ejemplo de estos pronombres es: aparta + se + lo→ apartárselo. Se utilizan con las formas del infinitivo, del gerundio, en modo imperativo, en perífrasis verbales y en el presente del subjuntivo. El análisis morfológico es el proceso de encontrar los elementos constitutivos (morfemas) de una palabra determinada; además muchas estrategias morfológicas se pueden representar mediante autómatas finitos (Martin y Jurafsky, 2009).

3.3.1 Transductores de estados finitos

Un transductor de estados finitos es un autómata de estados finitos que posee dos cintas en lugar de una. Se puede considerar como una extensión de un autómata de estados finitos que puede generar símbolos de salida. El autómata reconoce un símbolo si éste está presente en la cinta de entrada, aunque la manera más común de utilizarlos es imaginar que existe una cinta para la entrada y otra cinta para la salida. Desde este punto de vista, un transductor se dice que traduce el contenido de la cinta de entrada a la cinta de salida. Esta conversión se puede realizar de forma determinista o no determinista, generándose en este segundo caso una o más salidas por cada símbolo de entrada. Es posible el caso en el que el transductor no pueda producir ninguna salida para una cadena de entrada determinada; en este caso el transductor habrá rechazado la entrada. En general, podemos con-

siderar que un transductor establece una relación entre dos lenguajes formales cualesquiera, por lo que las reglas ortográficas pueden ser implementadas como transductores. Formalmente éstos pueden ser descritos mediante siete parámetros (Martin y Jurafsky, 2009):

$Q = \{q_0, q_1, \ldots, q_{N-1}\}$	conjunto finito de N estados
Σ	alfabeto finito de símbolos de entrada
Δ	alfabeto finito de símbolos de salida
$q_0 \in Q$	estado inicial
$F \subseteq Q$	conjunto de estdos finales
$\delta(q, w)$	función de transición entre estados
$\sigma(q, w)$	función de salida del conjunto de posibles cadenas de salida

(3.2)

Mientras que los autómatas de estados finitos son isomorfos a lenguajes regulares, los transductores de estados finitos son isomorfos a relaciones regulares. Éstas se definen como conjuntos de pares de cadenas, una extensión natural de los lenguajes regulares. Sobre los transductores pueden definirse operaciones de *inversión* y *composición*. El operador de *inversión* $T(T^{-1})$ intercambia las etiquetas de entrada y salida. Por tanto T asocian el alfabeto de entrada I al alfabeto de salida O, mientras que T^{-1} asocia O a I. El operador de *composición* actúa de la siguiente manera: Sean T_1 un transductor de I_1 a O_1 y T_2 un transductor de O_1 a O_2, entonces la composición $T_1 \circ T_2$ asocia de I_1 a O_2. La operación de proyección de un transductor de estados finitos es el autómata de estados finitos que se genera únicamente una parte de la relación. Esta proyección puede ser *principal* o *secundaria*. Las aplicaciones de los transductores secuenciales son múltiples, ya que traducen exactamente una cadena de entrada en una posible cadena de salida. Podemos encontrar numerosas aplicaciones de estos modelos en el procesamiento del lenguaje natural, tales como la representación de diccionarios, la compilación de reglas morfológicas y fonológicas, el establecimiento de restricciones sintácticas o el análisis morfológico (Mohri, 1996).

3.3.2 Morfología de dos niveles

La morfología de dos niveles en un modelo computacional desarrollado en los años 80 aplicable a la representación y análisis morfológico mediante transductores de estados finitos (Gelbukh, Sidorov y Velásquez, 2003). Es un modelo generalizable a cualquier lengua de forma sencilla y es válido tanto para la generación como para el análisis. Los dos niveles que se establecen son el nivel superficial (palabra a analizar) y el nivel léxico (representa el sistema léxico), por lo que evita almacenar distintas manifestaciones del mismo morfema, que han sido variadas por cambios

morfológicos. La diferencia más importante con la morfología generativa es que no existen estados intermedios ya que las reglas no ejecutan ninguna acción, sino que únicamente establecen relaciones entre los dos niveles. El reconocimiento de una palabra se reduce a encontrar una representación léxica válida que se corresponda con una forma de superficie. Los dos elementos constituyentes de la morfología de dos niveles son las reglas y el sistema léxico. Las primeras describen las diferencias entre los dos niveles. Las segundas generan el sistema léxico que define el conjunto de morfemas, clasificados según las posibles combinaciones de raíces y afijos. De esta manera la morfología de dos niveles define una correspondencia entre la palabra a analizar (nivel superficial) y la concatenación de morfemas (nivel léxico). Esta morfología puede ser utilizada tanto para el análisis morfológico como para la generación de palabras (Martin y Jurafsky, 2009).

3.3.3 Tokenización, segmentación de frases, corrección de errores

La *tokenización* es el proceso de reducir un texto a frases o palabras individuales tiene una gran importancia. No es tan sencillo como separar las palabras mediante el símbolo del espacio, ya que muchos lenguajes no lo utilizan. Además, pueden existir formas contraídas de expresiones (i.e. I'm en inglés es una contracción de I am) que deban ser separadas en varias palabras. Existen otros casos como nombres compuestos o las expresiones numéricas que pueden producir problemas (Martin y Jurafsky, 2009).

Por otra parte, el proceso de extracción de frases desde un texto se denomina *segmentación de frases* y es un proceso clave en el análisis inicial de un texto. Este proceso se basa fundamentalmente en los símbolos de puntuación. Pueden producir problemas algunas abreviaturas que lleven signos de puntuación (i.e. Mr.). En general los procesos de *tokenización* y *segmentación* se realizan mediante clasificadores binarios basados en reglas o en *machine learning*, que sean capaces de determinar si una frase es independiente o si pertenece a alguno de estos casos problemáticos (Martin y Jurafsky, 2009).

Con respecto a la corrección de errores de escritura, se puede entender que exista un determinado nivel de errores en un texto, ocasionado por diversas fuentes de error. Para el correcto funcionamiento de los algoritmos, es necesario corregir estos errores previamente; por tanto necesitamos conocer cuál es la palabra de nuestro diccionario más cercana a la palabra errónea. La aproximación básica a este problema es mediante la utilización de algoritmos de *minimización de la distancia de edición* (edit distance) (Wagner y Fischer, 1974); dicho de otra manera, encontrar el mínimo número de cambios producidos sobre la palabra errónea para

encontrar una palabra válida de nuestro diccionario. La palabra más cercana será aquella cuyo número total de cambios sea menor (Martin y Jurafsky, 2009).

3.4 Los N-gramas

En el análisis del lenguaje natural es muy importante la predicción de palabras; esto significa que dada una frase cualquiera nuestros modelos deben de ser capaces de calcular la probabilidad de que la siguiente palabra sea una determinada. Estos modelos estadísticos para el análisis de secuencias de palabras se conocen como *modelos de lenguaje*; calcular la probabilidad de la siguiente palabra, como veremos, está íntimamente relacionado con el cálculo de la probabilidad de una secuencia de palabras. El cálculo de probabilidades se va a fundamentar en el recuento de la aparición de casos particulares. En nuestro caso, se realizarán recuentos sobre un conjunto de grandes cantidades de texto denominado *corpus*. Algunos corpus muy conocidos con el *corpus de Brown*, o el *corpus de Switchboard*. En estos corpus la procedencia de los textos es de muy diversa índole, procediendo de fuentes tales como periódicos, artículos, fragmentos de libros, conversaciones transcritas, etc. (Martin y Jurafsky, 2009).

Continuaremos con el objetivo de calcular determinadas probabilidades sobre secuencias de textos; si tenemos una secuencia de textos a la que denominaremos h, ¿Cuál será la probabilidad de que la siguiente palabra de la secuencia sea w? Tendremos que calcular la probabilidad condicionada de w a la secuencia h, o lo que es lo mismo $P(w|h)$. Supongamos que la secuencia h es esta frase «hoy hace tanto frío que», ¿Cómo podemos calcular que la siguiente palabra w sea «nevará»? Formalmente tendríamos que calcula la siguiente probabilidad condicionada:

$$P(\text{nevará}|\text{hoy hace tanto frío que}) = \frac{P(\text{hoy hace tanto frío que nevará})}{P(\text{hoy hace tanto frío que})} \quad (3.3)$$

Por tanto deberíamos realizar un conteo para determinar el número de veces que aparece la frase «hoy hace tanto frío que» y otro conteo para determinar el número de veces que aparece la frase completada con la siguiente palabra «hoy hace tanto frío que nevará». Lo único que necesitamos es un corpus lo suficientemente amplio como para poder realizar estos conteos. Este método puede funcionar bien en algunos casos pero nos limita a realizar conteos únicamente con frases o secuencias de palabras que se encuentren en el corpus. Además, si quisiésemos averiguar la lista de todas las palabras más probables que pueden continuar a la frase, tendríamos que realizar un número muy elevado de conteos, recorriendo todas las opciones posibles. Esto sería totalmente inabordable desde un punto

de vista computacional. Necesitamos un método para estimar estas probabilidades que nos resuelva estos dos problemas. Formalizando matemáticamente este problema, denominaremos $P(w)$ a la probabilidad de aparición de la palabra w, por ejemplo, $P(\text{Marte})$ denomina a la probabilidad de aparición de la palabra «Marte» sobre el conjunto de todas las palabras de un determinado corpus. Una frase o secuencia de N palabras $w_1, w_2, \ldots, w_n$ será representada como w_1^n. Por tanto, la probabilidad de esa determinada secuencia de palabras vendrá denominada por $P(w_1, w_1, \ldots, w_n) = P(w_1^n)$. Si aplicamos la regla de la cadena en las probabilidades, tenemos que

$$P(w_1^n) = P(w_1)P(w_1)P(w_2|w_1)P(w_3|w_1^2)\ldots P(w_n|w_1^{n-1}) = \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \tag{3.4}$$

Sin embargo utilizar esta expresión que hemos utilizado gracias a la regla de la cadena no nos resulta demasiado útil; puede que determinadas combinaciones de palabras no aparezcan ya que el orden en el lenguaje es un factor determinante. En este punto es donde tenemos que introducir los N-gramas. La idea general de estos N-gramas es que podemos *aproximar* el cálculo de la probabilidad de la secuencia entera por tan solo la probabilidad de unas pocas n últimas palabras. El modelo *bigrama*, por ejemplo, aproximará la probabilidad condicional de una palabra a la frase anterior $P(w_n|w_1^{n-1})$ a la probabilidad condicional de la palabra con su palabra inmediatamente anterior $P(w_n|w_{n-1})$. De esta manera nos ahorraremos una gran cantidad de cálculos. Siguiendo con nuestro ejemplo, la probabilidad $P(\text{nevará}|\text{hoy hace tanto frío que})$ se podrá aproximar a la probabilidad $P(\text{nevará}|\text{que})$. Cuando utilizamos el modelo *bigrama*, realizamos la *suposición de Markov*, que consiste en afirmar que la probabilidad de aparición de una palabra depende únicamente de la palabra anterior. Realizando esta aproximación, el cálculo de las probabilidades queda de la siguiente manera

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1}) \tag{3.5}$$

Podríamos generalizar el modelo del *bigrama* al *trigrama*, considerando las dos palabras anteriores, y de la misma manera llegar al *N-grama*, que buscaría en las $N-1$ palabras previas. Sobre el modelo del *bigrama*, la probabilidad de una secuencia de palabras puede estimarse mediante el cálculo individual de cada uno de los pares de palabras:

$$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-1}) \tag{3.6}$$

¿Cómo podemos calcular la probabilidad de dos pares de palabras cualesquiera x e y? Utilizaremos las denominada *estimación de máxima probabilidad*, que consistiría en hace el conteo del número de veces que aparece el bigrama $C(xy)$ y normalizarlo dividiendo por el conteo total de todos los bigramas existentes que contienen como primera palabra x

$$P(x|y) = \frac{C(yx)}{\sum_w C(xw)} \tag{3.7}$$

Como la suma de todos los *bigramas* que comienzan con una palabra y tiene que ser igual al número total de *unigramas* con la palabra y, la ecuación anterior se simplifica de esta manera

$$P(x|y) = \frac{C(yx)}{C(y)} \tag{3.8}$$

que aplicado a nuestro modelo de bigrama en el que las palabras son w_n y w_{n-1} quedaría de la siguiente forma

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \tag{3.9}$$

En general, la probabilidad para un modelo de N-grama general sería la siguiente:

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})} \tag{3.10}$$

Será ahora muy sencillo calcular la probabilidad condicional de una secuencia de palabras cualquiera; siguiendo con el ejemplo anterior

$$P(\text{hace tanto frío que nevará}) = P(\text{tanto}|\text{hace})P(\text{frío}|\text{tanto})P(\text{que}|\text{frío})P(\text{nevará}|\text{que}) \tag{3.11}$$

Los modelos de n-gramas requieren de un entrenamiento; por ejemplo para el *bigrama* el entrenamiento consistiría en calcular mediante conteo la probabilidad condicionada de cada par de palabras distintas sobre un corpus, y almacenar estos resultados en una tabla. Por tanto, las probabilidades de los modelos N-gramas dependen del corpus que se ha utilizado para su entrenamiento. Debemos diferenciar entre corpus de entrenamiento y corpus de test; El corpus de test nos puede servir para evaluar diferentes arquitecturas de N-gramas en las que haya diferentes órdenes de N o en las que se hayan utilizado distintos tipos de algoritmos. Una vez entrenados los distintos modelos de N-gramas en el corpus de entrenamiento, podríamos seleccionar al mejor de ellos utilizándolos en el corpus de test (Martin y Jurafsky, 2009).

3.4.1 La Perplejidad

La *perplejidad* nos permite establecer una métrica de evaluación de un modelo basado en N-gramas. Este concepto se define mediante una función de probabilidad que el modelo asigna al corpus de test. Para un corpus de test definido como $W = w_1 w_2 \dots w_N$, la perplejidad es la probabilidad de la secuencia formada por las palabras del corpus de test normalizado por su número de palabras:

$$PP(W) = \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \dots w_{i-1})}} \tag{3.12}$$

En el caso de un *bigrama*, la perplejidad se definirá como

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_{i-1})}} \tag{3.13}$$

Podemos observar como cuanto mayor es la probabilidad de la secuencia de palabras del corpus de test, menor será su perplejidad, por tanto el hecho de minimizar la perplejidad es equivalente a aumentar la probabilidad del corpus de test. Los modelos de N-gramas deben ser construidos sin utilizar ningún tipo de información o conocimiento del corpus de test, ya que de lo contrario se puede generar una perplejidad artificialmente baja. Un *vocabulario cerrado* es aquel en que dado un diccionario, el corpus de test únicamente puede contener palabras que pertenezcan a este diccionario y por tanto todas las palabras son conocidas, no existiendo palabras desconocidas. Sin embargo pueden aparecer palabras desconocidas, que no se encuentran en el diccionario. Un vocabulario abierto será aquel en el que se puede modelizar las palabras desconocidas añadiendo una pseudopalabra *UNK* en el corpus de test. Observamos como un vocabulario cerrado, en el cual el vocabulario para el corpus de test haya sido preestablecido de antemano puede reducir enormemente la perplejidad (Martin y Jurafsky, 2009).

Es importante no mezclar las oraciones del corpus de prueba con el corpus de entrenamiento, ya que si nuestra secuencia de prueba forma parte del corpus de entrenamiento se le asignará una probabilidad artificialmente alta cuando se someta al corpus de test. Esta situación se denomina *entrenamiento en el corpus de test* y debe ser evitada ya que introduce un sesgo tanto en las probabilidades como en la perplejidad. Además puede ser útil introducir una fuente de datos adicional para aumentar el corpus de entrenamiento (*corpus extendido*) que mantenemos

fuera de nuestro corpus de entrenamiento. Existe un compromiso entre el tamaño del corpus de test y el corpus de entrenamiento: por un lado queremos que nuestro corpus de test sea lo más grande posible (esto repercute en que el corpus de entrenamiento no sea suficientemente representativo). Por otro lado, un corpus de entrenamiento demasiado grande nos reduce el poder estadístico del corpus de test e invalida los resultados obtenidos (Martin y Jurafsky, 2009).

3.4.2 El smoothing

Existe un importante problema con las técnicas utilizadas para entrenar los modelos basados en N-gramas: los datos que se utilizan en el proceso de entrenamiento son *dispersos*, es decir, de las múltiples combinaciones de probabilidad calculadas, muchas de ella resultan ser cero a causa de que todas las posibles combinaciones de palabras del corpus de entrenamiento son mucho más grandes que el propio corpus es decir, necesitaríamos un corpus inmensamente mayor para poder evaluar de forma correcta todas las posibilidades que, aunque menos probables y aparecen raramente, son válidas. Por tanto a causa del limitado tamaño de nuestro corpus de entrenamiento van a aparecer en la matriz de aprendizaje multitud de probabilidades iguales a cero en combinaciones de palabra que realmente no la tienen. A estas combinaciones factibles pero insólitas, nuestra técnica de entrenamiento les va a asignar una probabilidad igual a cero que, cuando se utilice para calcular probabilidades de frases en las que intervenga, producirán una probabilidad total igual a cero. Esto también sucederá en el cálculo de la *perplejidad*, pudiendo generarse valores de perplejidad iguales a cero si se utilizan alguna combinación con probabilidad cero. Necesitamos métodos para estimar y asignar una nueva probabilidad a todos aquellos casos cuya probabilidad ha sido calculada en cero. El término *smoothing* se utilizará para referirse a las distintas técnicas existentes para realizar esta labor. El origen del término viene del hecho de que lo que se va a hacer es coger una fracción de la probabilidad total y repartirla de alguna forma entre todos los conteos que han sido cero durante la fase de aprendizaje. Algunos métodos de *smoothing* son (Martin y Jurafsky, 2009):

1. *Smoothing* de Laplace: Consiste en sencillamente añadir uno a todos los conteos que han obtenido un valor igual a cero. Posteriormente se calculará la probabilidad normalizando al dividir por el número total de casos. De esta forma nunca existirá un bigrama con conteo igual a cero; por tanto ningún bigrama tendrá probabilidad igual a cero. La aplicación de esta técnica implica realizar un cambio en el proceso de normalización: si se ha añadido un valor extra a cada bigrama, el conteo total tendrá que aumentarse en un valor igual al número de palabras con el que se ha generado el bigrama.

La probabilidad para cada bigrama se calculará tal y como se muestra a continuación:

$$P_{\text{Laplace}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$$

Siendo V el número total del palabras del lexicón utilizado en la creación de bigrama. Los bigramas no contemplados en el modelo serán asignados con la probabilidad más baja existente en el bigrama, que será igual a la calculada con la fórmula anterior y un conteo de uno.

2. Descuento de Good-Turing: Se basa en el concepto de que para estimar el número de casos que nunca has encontrado se necesita información del número de casos que has encontrado una única vez.

3. Interpolación: Construye una interpolación lineal entre múltiples probabilidades, estimando las vecinas que sean cero.

4. Backoff: Es un método no lineal. En la estimación del n-grama se permite volver hacia atrás a través de historias cada vez más cortas. Produce cambios abruptos en la estimación de la probabilidad si se le añade más datos. Se utiliza en combinación con otros métodos.

3.5 Modelos de Markov

Los modelos de Markov fueron introducidos por el matemático ruso Andrey Andreyevich Markov (1856-1922). Markov fue estudiante de Pafnuty Chebyshev y trabajó entre otros en el campo de la teoría de números, el análisis y la teoría de la probabilidad. A partir de 1906, Markov publicó sus primeras obras en variables aleatorias dependientes del tiempo. A. A. Markov estudió la secuencia de 20.000 letras en el poema de A. S. Pushkin *Eugeny Onegin*, descubriendo que la probabilidad de vocal estacionaria es $p = 0{,}432$, que la probabilidad de una vocal después de una vocal es $P_1 = 0{,}128$, y que la probabilidad de una vocal después de una estafa sonant es $P_2 = 0{,}663$ (Basharin, Langville y Naumov, 2004). El término *cadena de Markov* , en este trabajo, también conocido como *modelo de Markov* para esta clase de procedimientos estocásticos se utilizó por primera vez en 1926 en una publicación del matemático Ruso Sergey Natanovich Bernstein (Nierhaus, 2009).

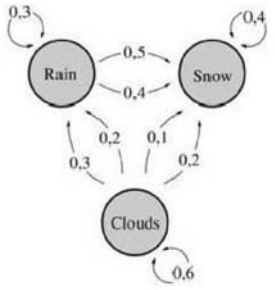
3.5.1 Fundamento teórico

Los procesos estocásticos se utilizan para describir una secuencia de eventos aleatorios dependientes del parámetro de tiempo (t). El conjunto de eventos se llama *espacio de estado*, mientras que el conjunto de parámetros se conoce como el *espacio de parámetro*. Si un proceso estocástico consiste en un número contable de estados, entonces también puede ser referido como una cadena estocástica. En una cadena estocástica, cada tiempo discreto t tiene una variable aleatoria X. En la cadena de Markov, siendo un tipo especial de cadena estocástica, la probabilidad del futuro estado X_{t+1}, es decir, la variable aleatoria X en el momento $t+1$, depende del estado actual X_t. Para los tiempos dados t_m y t_{m+1}, esta probabilidad es

$$P(X_{t_{m+1}} = j | X_{t_m} = i) = p_{ij}(t_m, t_{m+1}) \tag{3.14}$$

Esta expresión indica la probabilidad de transición del estado $X_{tm} = i$ en un momento dado T_m al estado $X_{tm+1} = j$ (Bronstein y col., 2012).

Una cadena de Markov puede representarse mediante un gráfico de transición de estado, o por una matriz de transición. La siguiente figura muestra las probabilidades de transición que se pueden encontrar en la matriz de transición P representada como una tabla. La suma de todas las probabilidades de transición en cada Estado debe ser igual a 1. A partir de un estado en particular, se pueden determinar las probabilidades para un estado futuro. Estas probabilidades se calculan con la fórmula $p(t+k) = p(t)P^k$, donde $p(t)$ representa el estado inicial, k es el número de transiciones de estado y P la matriz de transición (Bronstein y col., 2012).



Tomorrow's Weather

Today's Weather	Rain	Snow	Clouds
Rain	0,3	0,5	0,2
Snow	0,4	0,4	0,2
Clouds	0,3	0,1	0,6

Figura 3.4: Ejemplo de una cadena de Markov de primer orden (Nierhaus, 2009, pág. 68).

Si se utiliza más de un evento pasado en el cálculo de las probabilidades de transición, se denomina proceso de Markov de orden superior, el orden que indica el número de eventos pasados que son relevantes para las probabilidades de transición. En consecuencia, en un modelo de Markov basado en los tonos de tono de un corpus melódico, la secuencia de salida se aproximará más y más a la estructura del corpus con el aumento de las órdenes de Markov. Si una secuencia particular de longitud n no se produce en el corpus, entonces esta secuencia no aparecerá en el material producido en el correspondiente análisis de Markov y la generación de enésimo orden tampoco. Una posible solución a este problema es ofrecida por las técnicas de *smoothing* que utilizan probabilidades de transición de orden inferior para la generación de probabilidades de transición de orden superior. En este procedimiento, las probabilidades de transición faltantes para secuencias insuficientes de orden enésimo pueden adquirirse por interpolación con órdenes inferiores $n-1$, $n-2$, etc. En un símbolo dado en el punto actual, las probabilidades de transición calculadas desde el corpus obtienen diferentes pesos para las secuencias anteriores de distinta longitud. En general, los modelos de Markov solo pueden ocupar un número finito de estados y, por lo tanto, pueden representarse mediante autómatas finitos y mediante de una representación gráfica. Esta posibilidad es especialmente interesante en los casos en que no todos los campos de la matriz de transición necesitan ser ocupados (Nierhaus, 2009).

3.5.2 Modelos ocultos de Markov

En los modelos ocultos de Markov (HMM), las secuencias de los símbolos de salida observables de un modelo de Markov son visibles, pero sus estados internos y las transiciones de estado no lo son. Veamos el siguiente ejemplo: una agencia de noticias recibe su información sobre eventos políticos en un país extranjero de un corresponsal en diferentes momentos del día. En su trabajo, este corresponsal es, entre otras cosas, influenciado por la situación meteorológica. Así, por ejemplo, si el sol brilla, le gusta levantarse temprano y, por lo tanto, envía su informe antes del desayuno. Por otro lado, si está lloviendo, le gusta dormir un poco más y, por lo tanto, no comienza su rutina diaria antes de tomar algunas tazas de té fuerte. En consecuencia, dependiendo de la hora en que llegue el informe, la agencia de noticias también puede hacer inferencias sobre la situación del clima en el país extranjero. Pero, debido a que este no es el único factor determinante para la disciplina de trabajo del corresponsal, el momento en que llegan los informes puede sugerir una situación climática particular. Entonces, en analogía con un HMM, los tiempos específicos de llegada de los informes se pueden ver como las secuencias de los símbolos de salida observables del HMM generados por las probabilidades de emisión de los estados ocultos, la secuencia probable subyacente de

diferentes situaciones climáticas. Los modelos ocultos de Markov pueden entregar distribuciones continuas y discretas de probabilidades de emisión. Sin embargo, en la composición algorítmica, los modelos continuos son de poca importancia ya que las emisiones observadas son en la mayoría de los casos valores con parámetros cuantificados. Un modelo oculto de Markov, por tanto, representa un proceso estocástico que está acoplado, debido a las probabilidades de transición de los estados en el modelo de Markov y las probabilidades de emisión dependientes del estado de los eventos observados. Las siguientes indicaciones y símbolos se utilizan para la descripción formal de un HMM (Nierhaus, 2009; Knab, 2000).

N	Número de estados en el modelo de Markov
$\{S_1, \cdots, S_N\}$	Conjunto de estados
$\pi = \{\pi_1, \cdots, \pi_N\}$	Vector de probabilidades iniciales para cada estado
$A = \{a_{ij}\}$	Probabilidades de transición en el MM de un estado a otro
M	Número de símbolos de salida observables del HM
$\{v_1, \cdots, v_M\}$	Conjunto de símbolos de salida
$B = \{b_{jm}\}$	Probabilidades de emisión de un símbolo de salida en un determinado estado
T	Longitud de la secuancia de salida
$O = O_1 \cdots O_T$	Secuencia de símbolos de salida con $O_t \in \{v_1, \cdots, v_M\}$
$Q = q_1 \cdots q_T$	Secuencia de secuancias de estado en el MM con la salida O y $q_t \in \{1, \cdots T\}$

(3.15)

Para responder preguntas esenciales dentro de un modelo oculto de Markov, se aplican principalmente tres algoritmos (Nierhaus, 2009):

- El algoritmo *forward* (directo) calcula las probabilidades para la aparición de una secuencia observable particular, donde se conocen los parámetros (probabilidades de transición y observación, así como la distribución del estado inicial) del HMM.
- El algoritmo de *Viterbi* calcula la secuencia más probable de estados ocultos, llamada ruta de Viterbi, sobre la base de una secuencia observable dada.
- El algoritmo de *Baum-Welch* se aplica para encontrar los parámetros más probables de un HMM sobre la base de una secuencia observable dada.

Capítulo 4

Tecnologías del proyecto

A continuación realizaremos una breve descripción de las tecnologías utilizadas en nuestra investigación, agrupándolas en tecnologías de programación básica, tecnologías para el procesamiento del lenguaje natural y tecnologías para el análisis musicológico.

4.1 Programación básica

A continuación describiremos las tecnologías de programación básica utilizadas para realizar la implementación computacional de nuestro sistema.

4.1.1 Python

Python es un lenguaje de programación creado en la década de 1990 por Guido van Rossum en el Centro para las Matemáticas y la Informática (CWI, *Centrum Wiskunde and Informatica*), en los Países Bajos. A causa de u simplicidad y elegancia, ha ganado popularidad entre programadores de manera muy rápida, y ha sido adoptado por grandes empresas, como Google Inc. Comparado con otros lenguajes de programación, como Java y C++, Python es particularmente útil para aprender programación. Python ofrece un sistema conceptualmente muy simple para la ejecución de sentencias. Como resultado, Python resulta fácil de apren-

der y dominar completamente. En términos de funcionalidades, Python es tan poderoso como Java y C++. y además, existe una gran variedad de bibliotecas de código abierto que enriquecen el poder de Python, haciendo que sea una opción muy conveniente en el ámbito de muchas aplicaciones, como las bibliotecas *numpy* o *scipy*, que ofrecen formas muy rápidas de realizar cálculos científicos utilizando Python. Python es un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma. Se encuentra administrado por la *Python Software Foundation* y posee una licencia de código abierto, denominada *Python Software Foundation License*, compatible con la Licencia Pública General de GNU a partir de la versión 2.1.1. (Zhang, 2015).

4.1.2 PyScripter

PyScripter es un entorno de desarrollo integrado (IDE) de Python para software libre y de código abierto para Windows. Está construido en Object Pascal y Python. Originalmente comenzó como un IDE ligero diseñado para servir al propósito de proporcionar una solución de scripting sólida para las aplicaciones de Delphi. Con el tiempo, se ha convertido en un IDE de Python independiente con todas las funciones. Está construido en Delphi usando P4D y es extensible usando scripts de Python. Al estar construido en un lenguaje compilado, lo hace bastante ligero en comparación con algunos de los otros IDE. Actualmente, solo está disponible para los sistemas operativos *Microsoft Windows* (*PyScripter* 2019). Entre sus características incorpora:

- Editor de resaltado de sintaxis
- Intérprete de Python integrado
- Depurador de Python integrado
- Vistas del editor
- Explorador de archivos
- Pruebas unitarias integradas

4.2 Tecnologías para ontologías

4.2.1 OWL

El lenguaje OWL (*Web Ontology Language*) lo constituyen una familia de lenguajes de representación de conocimiento orientados a la creación de ontologías, que son una forma de describir taxonomías y redes de clasificación. Permiten definir la estructura del conocimiento para cualquier dominio. Las ontologías, a diferencia de las jerarquías de clase, están destinadas a representar la información en internet, permitiendo la evolución constante y flexible de la misma a partir de todo tipo de fuentes de datos. El lenguaje OWL se caracteriza por su semántica formal, basada en el estándar XML del *World Wide Web Consortium* (W3C) orientado a objetos, denominado *Resource Description Framework* RDF). La familia OWL contiene muchas especies, serializaciones, sintaxis y especificaciones con nombres similares (*Web Ontology Language* 2019).

4.2.2 Protégé

Protégé es una plataforma gratuita de código abierto que proporciona un conjunto de herramientas para construir modelos de dominio y aplicaciones basadas en el conocimiento mediante ontologías, desarrollado por la Universidad de Stanford en colaboración con la Universidad de Manchester. Constituye, además, una gran comunidad de usuarios en continuo crecimiento con más de 300,000 miembros registrados. El programa está desarrollado sobre Java y utiliza Swing para la interfaz de usuario (Stevens, 2016).

4.3 Procesamiento del lenguaje natural

4.3.1 NLTK

El kit de herramientas de lenguaje natural (*Natural Language Toolkit*), o más comúnmente NLTK, es un conjunto de bibliotecas y programas para el procesamiento de lenguaje natural simbólico (PNL) escrito en el lenguaje de programación python. Ha sido desarrollado por Steven Bird y Edward Loper en el Departamento de Informática y Ciencia de la Información de la Universidad de Pennsylvania. El kit incluye demostraciones gráficas así como datos de muestra y distintos corpus. Está destinado a apoyar la investigación y la enseñanza en el Procesamiento del Lenguaje Natural o en áreas relacionadas como por ejemplo la lingüística empírica, la ciencia cognitiva, la inteligencia artificial ola recuperación de información

y el aprendizaje automático. NLTK se ha utilizado con éxito utilizándola como herramienta de enseñanza y como plataforma sobre la que diseñar prototipos y construir sistemas de investigación. NLTK proporciona diversas funciones de clasificación, tokenización, derivación, etiquetado, análisis y razonamiento semántico (Bird y Loper, 2019).

Aunque python ya contiene implementadas la mayoría de las funcionalidades necesarias para realizar tareas simples de PNL, todavía no es lo suficientemente potente para la mayoría de las tareas PNL estándar. Aquí es donde interviene NLTK, gracias a su colección de módulos y corpus, lanzada bajo una licencia de código abierto, que permite a los estudiantes aprender y realizar investigaciones en la PNL. La ventaja más importante de usar la NLTK Es que es totalmente autónomo. No sólo proporciona funciones convenientes que se pueden usar como bloques de construcción para tareas comunes de PNL, sino que también proporciona versiones preprocesadas de corpus estándar utilizados en la literatura y cursos de PNL (Madnani, 2007).

4.4 Análisis musicológico

4.4.1 Music21

Music21 es un conjunto de herramientas orientado a objetos basado en Python para musicología asistida por ordenador. music21 es capaz de resolver cuestiones abiertas en la musicología moderna gracias a la utilización de computadoras, como por ejemplo estudiar grandes conjuntos de datos de música, generar ejemplos musicales, enseñar fundamentos de la teoría musical, editar la notación musical, estudiar música o componer música, tanto de manera algorítmica como directa, siempre desde un punto de vista simbólico. El sistema existe desde 2008 y está en constante crecimiento y expansión gracias a Michael Cuthbert, Christopher Ariza, Benjamin Hogue y Josiah Wolf Oberholtzer, en el MIT. El framework recibe su nombre del hecho de que fue principalmente creado en el MIT desde la Sección de Artes de Música y Teatro, designada con el número 21M (Cuthbert y Ariza, 2010).

Music21 importa partituras de *MIDI*, *MusicXML*, *Humdrum*, *abc*, *MuseData*, *Noteworthy Composer* y audio monofónico. Contiene soporte para extraer características de la música (feature extraction) como la altura de las notas, los acordes, la línea del bajo y ornamentos. Es capaz también de extraer las letras de las canciones, determinar algorítmicamente la tonalidad de la obra, extraer los acordes

y sus grados tonales y proporcionar su cifrado en distintos estilos como el barroco o el americano (Cuthbert y col., 2011).

Por último, music21 incorpora un importante corpus de composiciones en formato MusicXML entre las que se incluyen:

- Los 438 corales de J. S. Bach.
- Todos los madrigales de Claudio Monteverdi.
- 1300 composiciones contrapuntísticas de Giovanni Palestrina.
- *Essen Folksong Collection*.

4.4.2 MusicXML

MusicXML es un lenguaje de intercambio de música basado en XML. Está pensado específicamente para representar la notación musical occidental común desde el siglo XVII en adelante, incluida la música clásica y la popular. El lenguaje está diseñado para ser extensible a la cobertura futura de la música antigua y las necesidades de notación menos estándar de los puntajes de los siglos veinte y veintiuno. MusicXML está diseñado para admitir el intercambio entre aplicaciones de notación musical. MusicXML no pretende reemplazar otros formatos que están optimizados para aplicaciones musicales específicas, sino que permite compartir datos musicales entre aplicaciones. El objetivo del desarrollo es apoyar el intercambio con cualquier programa musical para notación occidental con un formato de datos informáticos publicados (Good, 2001b).

El formato es abierto, está totalmente documentado y se puede utilizar libremente en virtud del Acuerdo de especificación final de la comunidad del W3C. Fue diseñado por Michael Good y desarrollado inicialmente por Recordare LLC, derivó de varios conceptos clave de los formatos académicos existentes (como MuseData de Walter Hewlett y Humdrum de David Huron). El desarrollo de MusicXML fue gestionado por *MakeMusic* tras la adquisición de Recordare por la empresa en 2011. El desarrollo de MusicXML se transfirió al Grupo de la Comunidad de la Notación Musical del W3C en julio de 2015 (Good, 2001a).

La versión 1.0 se lanzó en enero de 2004. La versión 1.1 se lanzó en mayo de 2005 con un soporte de formato mejorado. La versión 2.0 se lanzó en junio de 2007 e incluía un formato comprimido estándar. Todas estas versiones se definieron mediante una serie de *definiciones de tipo de documento* (DTD). Una implementación de la definición de esquema XML (XSD) de la Versión 2.0 que se lanzó en

septiembre de 2008. La Versión 3.0 se lanzó en agosto de 2011 con un soporte mejorado para instrumentos virtuales VST, tanto en las versiones DTD como XSD. La versión 3.1 se lanzó en diciembre de 2017 con soporte mejorado para el diseño de fuente de música estándar (SMuFL). Los DTD y los XSD de MusicXML se pueden redistribuir libremente en virtud del Acuerdo de especificación final de la comunidad del W3C (*musicXML* 2019).

4.4.3 MuseScore

MuseScore es un programa de notación musical de software libre, publicado bajo la licencia pública general de GNU, para *Linux*, *Mac OS X* y *Microsoft Windows*. Ofrece soporte completo para reproducir partituras e importar o exportar MusicXML y archivos MIDI estándar, pudiendo generar también documentos en formato PDF, SVG o PNG, o exportando la música a formato *LilyPond*. El programa tiene una interfaz de usuario sencilla, con una rápida entrada de notas en edición similar al ingreso rápido de notas que tienen otros programas comerciales de notación musical, como *Finale* y *Sibelius*. Está respaldado por una gran comunidad de usuarios que mantienen una amplia base de datos de canciones en formato MusicXML (*Musescore.org* 2019).

4.5 Resumen de las tecnologías empleadas

En la siguiente tabla podemos encontrar todas las tecnologías utilizadas en el proyecto:

Tabla 4.1: Tecnologías utilizadas en el proyecto

Tipo	Software	Uso en el proyecto
Framework	NLTK v3.4.1	Creación del modelo de lenguaje musical.Tokenización. Construcción de bigramas y trigramas.
	music21 Toolkit v5.5	Parsing de archivos en MusicXML. Codificación y detección de acordes. Extracción de las figuras rítmicas. Escritura de la partitura en musicXML
	pickle	Serialización de objetos en python.
Lenguaje	python v3.7	Lenguaje de programación del proyecto.
	MusicXML v3.0	Formato abierto de notación musical basado en XML.
IDE	PyScripter v3.6	Entorno integrado de desarrollo opensource para python.
Control Versiones	Apache Subversion 1.11.1	Gestor de versiones en local durante la fase de desarrollo.
	GitHub	Repositorio para alojar el proyecto final.
Corpus	music21 Corpus	Corpus de obras opensource de distintos compositores: J. S. Bach, Monteverdi, Essen Folksong Collection
	musescore.org	Web con gran cantidad de obras en formato MusicXML. Recolección de gran cantidad de estándares de música jazz.
Edición partituras	MuseScore v3.0.5	Visualización y reproducción del material musical generado

La extracción de características musicales, tanto de índole armónico como rítmico se realizará mediante el framework music21. Con él, podremos obtener la secuencia de acordes, en cifrado romano, de cualquier composición de nuestro corpus, reduciendo una obra a un conjunto de símbolos que codifican las secuencias armónicas de la misma. El framework nos proporcionará herramientas para la detección

de la armadura y hacer este proceso válido para cualquiera que sea la tonalidad de la obra. También podremos representar simbólicamente, y de manera unívoca la duración de cada nota mediante un coeficiente que expresará su duración de manera numérica. El toolkit music21 nos permitirá también la generación de material musical y su exportación a formato MusicXML, válido para multitud de editores de partituras, entre los cuales destaca MuseScore. Gracias a este último, seremos capaces de visualizar y escuchar la obra generada. La generación y suavizado de los bigramas y trigramas se realizará mediante el toolkit NLTK, estudiado en varias de las asignaturas del máster, que proporciona funcionalidades para el cálculo y representación de n-gramas como diccionarios que representan matrices de tipo dispersa. La implementación informática se realizará en python, bajo el IDE PyScripter.

Capítulo 5

Diseño de una Ontología musical

En este capítulo describiremos el proceso de desarrollo de la ontología musical utilizada en la presente investigación para la implementación de estructuras de control formal subyacentes al modelo de composición musical markoviano.

El discurso musical se articula en torno a la construcción de grandes unidades formales mediante la sucesión de unidades más cortas. Las secciones de las obras se componen de frases, las frases a su vez se articulan jerárquicamente mediante los encadenamientos armónicos finales, denominados cadencias. Todo este engranaje conforma lo que se conoce como armonía funcional y constituye los pilares fundamentales de las estructuras formales.

Sin embargo, las producciones musicales obtenidas mediante modelos de Markov carecen de esta jerarquía estructural: muestran una tendencia a divagar de forma un tanto errática, sin la direccionalidad que una estructura subyacente preestablecida debería conferirles. El diseño de esta ontología, para su posterior implementación en el sistema compositivo, pretende dotar a las composiciones generadas por nuestro sistema markoviano de la coherencia formal y armónica que la música compuesta por humanos posee.

5.1 Metodologías para diseño de ontologías

Para el diseño de la ontología se ha utilizado la metodología *Methontology*, desarrollada por Fernández-López, Gómez-Pérez y Juristo (1997) el Laboratorio de Inteligencia Artificial de la Universidad Politécnica de Madrid. La metodología propone las siguientes actividades: planificación, adquisición del conocimiento, documentación y evaluación, a través de una serie de estados que son: especificación, conceptualización, formalización, integración, implementación y mantenimiento. Este esquema puede verse reflejado en la siguiente figura.

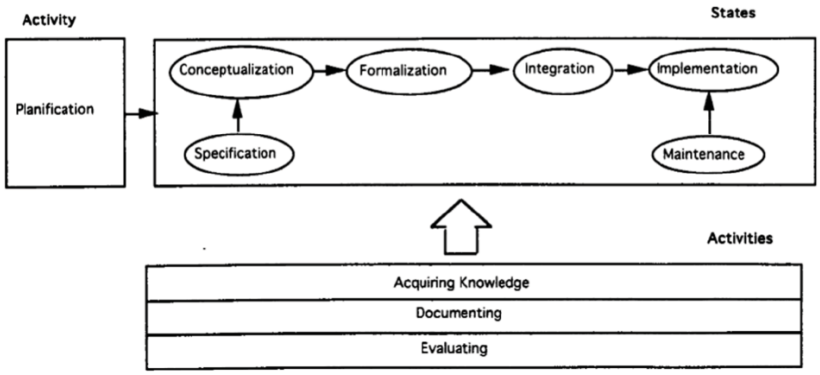


Figura 5.1: Estados y actividades en la metodología Methontology (Fernández-López, Gómez-Pérez y Juristo, 1997, pág. 35).

5.2 Sistema experto

Para el diseño de la ontología musical, así como para el desarrollo de la presente investigación se ha realizado la elicitación de conocimiento experto a partir del análisis y conocimiento de las estructuras formales y armónicas de la música culta occidental, también denominada *música clásica*.

Para la adquisición de conocimiento se ha aprovechado el nuestro propio. El autor de la presente investigación ha realizado estudios Superiores de Música en la especialidad de Composición y desarrolla una actividad compositiva en paralelo a la actividad investigadora. Estamos por tanto en el caso en el que nosotros mismos somos la fuente experta en conocimiento musical.

Con independencia de esto, se ha refrendado el conocimiento experto mediante la consulta de fuentes bibliográficas de referencia en el estudio musical como son el

Tratado de la forma musical de Bas y Lamuraglia (1977) y el *Tradato de Armonía* de Zamacois (2002).

Se han realizado una serie de mapas conceptuales como resultado de la elicitación del dicho conocimiento, en los que se han expuesto, de manera organizada, los principales aspectos formales y armónicos que nuestro sistema experto tiene que cumplir.

5.3 Construcción de la ontología

- **Especificación:** El dominio de la ontología es la representación simbólica de la música. Utilizaremos esta ontología para controlar las estructuras armónicas y formales del material musical generado por nuestro modelo. El tipo de la ontología será de dominio, con presencia de fuertes conceptualizations del conocimiento, así como un gran número de reglas por lo que se puede considerar como una *ontología pesada*.
- **Adquisición de conocimiento:** La adquisición del conocimiento se realizará mediante la elicitación del conocimiento experto anteriormente explicada.
- **Conceptualización:** La ontología tiene que incorporar el siguiente glosario de términos (GT): los conceptos de nota y grado musical, acorde, cadencia, frase, forma así como una tipología de los mismos que encaje en teoría de la armonía tonal funcional. Además necesitamos otros conceptos relativos a la teoría musical como el compás, la tonalidad, el tempo, etc.
- **Integración:** Con el fin de acelerar la creación de la ontología, utilizaremos como punto de partida una ya existente, en concreto usaremos *MusicOWL*[1], una ontología musical con licencia *GNU General Public License* que incluye muchas clases, propiedades y reglas relativas al dominio musical predefinidas. Incluiremos nuestra conceptualización utilizando como punto de partida dicha ontología. La selección de dicha ontología, y no otra distinta, para esta fase de integración será discutida en la siguiente sección.

[1] MusicOWL puede ser descargada de `https://github.com/jimjonesbr/musicowl`.

5.4 Revisión de ontologías musicales existentes

A continuación enumeraremos distintas ontologías musicales preexistentes, enumerando sus propiedades y valorando su potencial uso como ontología de partida para la fase de integración, poniendo de manifiesto la necesidad de implementar una extensión ontológica que cumpla con nuestros requisitos.

5.4.1 MusicOntology

MusicOntology proporciona los principales conceptos y propiedades para describir la música comercial, por ejemplo el artista, los álbumes o canciones, dentro de la web semántica (Raimond y col., 2007). Ha sido desarrollada en 2006 por Yves Raimond, Thomas Gängler, Frédérick Giasson, Kurt Jacobson, George Fazekas, Simon Reinhardt y Alexandre Passant. Utiliza el framework RDF / OWL y se utiliza ampliamente en varias aplicaciones, incluidos los recursos disponibles en *DBTune*[2]. Su última versión puede ser descargada de `http://purl.org/ontology/mo/`

Esta ontología es muy completa, sin embargo está orientada a la clasificación y distribución comercial de la música, para la correcta organización de metadatos y clasificación de los géneros musicales. No contempla aspectos técnicos de la música, como la definición de conceptos como acordes, ritmos, escalas, etc. que sí serían de utilidad para nuestro propósito.

5.4.2 OMRAS2 Chord Ontology

Publicada en 2007 por los autores Christopher Sutton, Yves Raimond, Matthias Mauch, esta ontología proporciona un vocabulario común y versátil para describir acordes y secuencias de acordes en RDF. La ontología utiliza un modelo basado en la notación original de Christopher Harte para los acordes, con algunas modificaciones para que se adapte mejor al dominio y permita una mayor flexibilidad a la hora de describir los acordes. Puede se descargada de `http://motools.sourceforge.net/chord_draft_1/chord.html`.

Esta ontología se acerca un poco más a nuestras necesidades, sin embargo es demasiado sencilla y nos requeriría de demasiadas implementaciones.

[2] *DBTune* recoge un gran número de servidores, permitiendo el acceso a estructuras de datos relacionados con la música basados en estándares Web cono el RDF. Para más información véase `http://dbtune.org/`

5.4.3 The Temperament Ontology

Diseñada en 2008 por Gyorgy Fazekas, Dan Tidhar y el *Centre for Digital Music Queen Mary University of London*, esta ontología se puede utilizar para describir con gran detalle la afinación de un instrumento. Se entiende su uso como un complemento de la Ontología Musical, permitiendo por ejemplo describir el sistema de afinación utilizado en una determinada grabación comercial. puede descargarse de `http://dbtune.org/onto/tm/doc/temperament.html`.

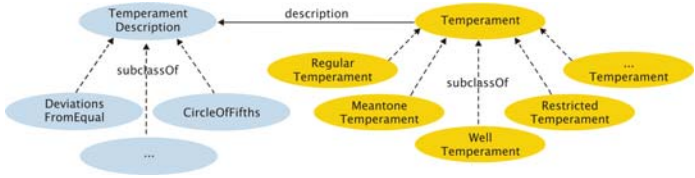


Figura 5.2: Vista general de The Temperament Ontology (Fazekas y Tidhar, 2009).

5.4.4 Music Notation Ontology

Desarrollada en 2016 por Fayçal Hamdi y Samira Si-said Cherfi en el *Conservatoire National des Arts et Métiers* de París, esta ontología proporciona una descripción simple para la armonía, haciendo énfasis en la nota musical, el silencio, la voz e incluso la letra. Puede descargarse de `http://cedric.cnam.fr/isid/ontologies/files/MusicNote.html`

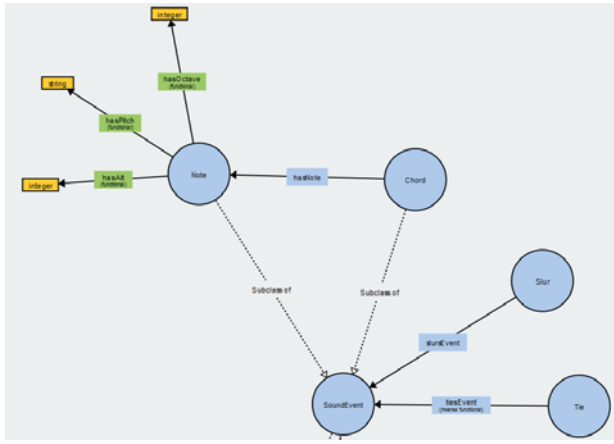


Figura 5.3: Ejemplo en WebVOWL de parte de la Music Notation Ontology (Elaboración propia).

5.4.5 MusicOWL

MusicOWL, diseñada por Jim Jones y Diego de Siqueira Braga (Jones y col., 2017) es una ontología para describir formalmente todos los contenidos de las partituras de música occidental. A diferencia de otrás ontologías, MusicOWL proporciona un completo vocabulario para posibilitar la notación simbólica de partituras musicales, cubriendo muchos aspectos de la información relevante relacionada con la música, como por ejemplo la melodía, las dinámicas, la tonalidad, etc.

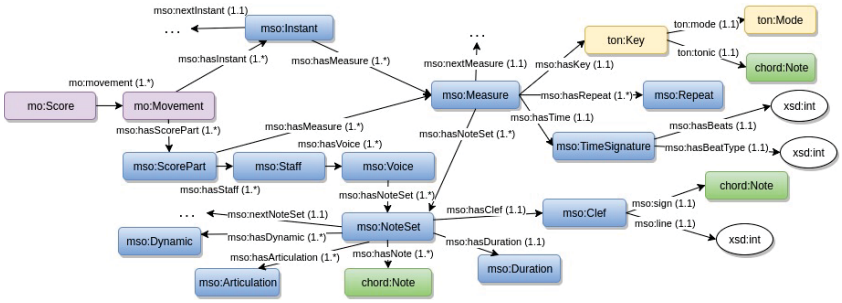


Figura 5.4: Vista general de MusicOWL (Jones y col., 2017, pág. 1224).

Encontramos que esta ontología es la más relevante y completa de las estudiadas, y nos confiere una buena aproximación a los propósitos de nuestro trabajo de investigación. Puede ser descargada de en OWL desde `https://github.com/jimjonesbr/musicowl`.

5.5 Implementación en Protégé

Utilizaremos el lenguaje *OWL* y el software *protégé* para realizar la implementación de nuestra ontología, a modo de extensión sobre la ontología preexistente MusicOWL, basándonos en los conceptos y relaciones jerárquicas de los mismos obtenidos mediante el análisis procedente del conocimiento experto. OWL es un lenguaje ampliamente utilizado en la web semántica y soportado por protégé. A continuación veremos los resultados de la implementación de nuestra ontología.

La ontología puede consultarse en la siguiente url:
`https://github.com/BrianComposer/MusicOWLExtension`

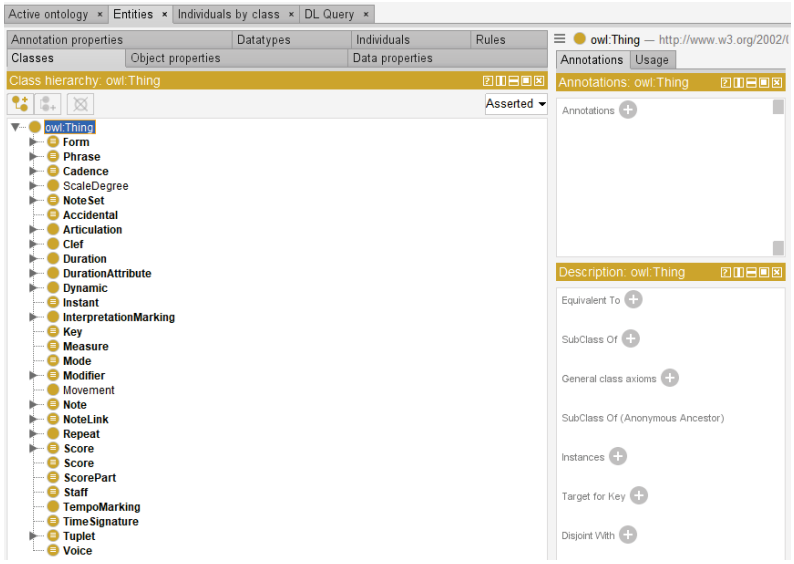


Figura 5.5: Implementación de la ontología en *protégé*.

5.5.1 Implementación de las cadencias armónicas

A continuación mostraremos la relación entre los acordes, los grados de la escala (I, II, III, IV, V, VI, VII) y los diferentes tipos de cadencias básicas que han sido implementadas en la ontología (perfecta, semicadencia, rota, o plagal).

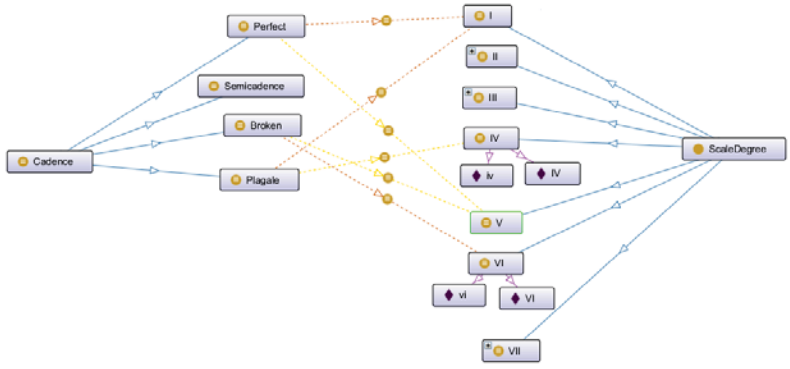


Figura 5.6: Tipos de cadencias y su relación con los acordes de los grados musicales.

Podemos ejemplificar, en *pseudocódigo*, algunas de las reglas que han sido definidas en nuestra ontología a través de Protégé para la definición de estas cadencias:

```
cadence is Perfect if:
    cadence is subClass of Cadence
    and
    cadence.hasChord is exactly 2 ChordDegree
    and
    cadence.firstChord is exactly V
    and
    cadence.secondChord is exactly I


cadence is Plagale if:
    cadence is subClass of Cadence
    and
    cadence.hasChord is exactly 2 ChordDegree
    and
    cadence.firstChord is exactly IV
    and
    cadence.secondChord is exactly I


cadence is Broken if:
    cadence is subClass of Cadence
    and
    cadence.hasChord is exactly 2 ChordDegree
    and
    cadence.firstChord is exactly V
    and
    cadence.secondChord is exactly VI


cadence is Semicadence if:
    cadence is subClass of Cadence
    and
    cadence.hasChord is exactly 2 ChordDegree
    and
    cadence.firstChord is (I or II or IV or VI)
    and
    cadence.secondChord is exactly V
```

5.5.2 Implementación de la forma musical

Las formas musicales que utilizaremos se clasificarán en forma binaria (AB)y forma ternaria (ABA). La forma binaria incluye dos frases (una frase de tónica y otra de dominante); la forma ternaria incluye tres frases (frase tónica, frase dominante, frase tónica conclusiva). Una frase requiere de un mínimo de 4 compases y un máximo de 8 y ha de empezar por un grado en concreto y finalizar con una cadencia posible. De esta manera la frase tónica comienza con el primer grado (I) y finaliza con semicadencia o con cadencia rota; la frase dominante comienza con el quinto grado (V) y finaliza con cadencia perfecta o plagal; la frase tónica conclusiva comienza con el primer grado y acaba con cadencia perfecta.

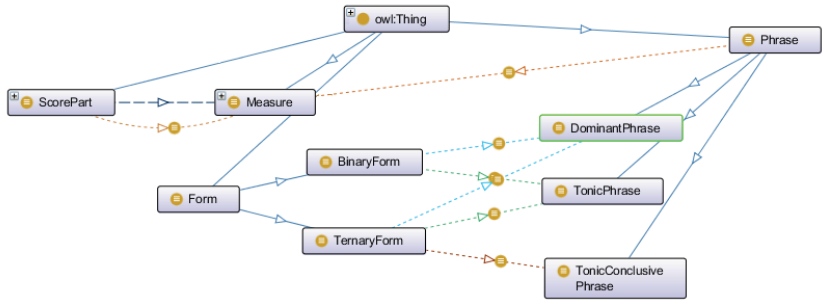


Figura 5.7: Formas binaria y ternaria y su relación con las distintas cadencias.

A continuación mostraremos, nuevamente en pseudocódigo, las reglas implementadas para los tres tipos básicos de frases musicales con las que se construirán las formas binarias y ternarias.

```
phrase is TonicPhrase if:
    phrase is subClass of Phrase
    and
    phrase.hasMeasure >= 4 Measures and phrase.hasMeasure <= 8 Measures
    and
    phrase.hasCadence is exactly 1 Cadence
    and
    phrase.firstChord is exactly I
    and
    phrase.hasCadence is (Broken or Semicadence)


phrase is TonicConclusivePhrase if:
    phrase is subClass of Phrase
```

```
    and
    phrase.hasMeasure >= 4 Measures and phrase.hasMeasure <= 8 Measures
    and
    phrase.hasCadence is exactly 1 Cadence
    and
    phrase.firstChord is exactly I
    and
    phrase.hasCadence is exactly Perfect


phrase is DominantPhrase if:
    phrase is subClass of Phrase
    and
    phrase.hasMeasure >= 4 Measures and phrase.hasMeasure <= 8 Measures
    and
    phrase.hasCadence is exactly 1 Cadence
    and
    phrase.firstChord is exactly V
    and
    phrase.hasCadence is (Perfect or Plagale)
```

Por último, el pseudocódigo de las formas binarias y ternarias

```
form is BinaryForm if:
    form is subClass of Form
    and
    form.hasPhrase is exactly 2 Phrase
    and
    phrase.firstPhrase is exactly TonicPhrase
    and
    phrase.secondPhrase is exactly DominantPhrase


form is TernaryForm if:
    form is subClass of Form
    and
    form.hasPhrase is exactly 3 Phrase
    and
    phrase.firstPhrase is exactly TonicPhrase
    and
    phrase.secondPhrase is exactly DominantPhrase
    and
    phrase.thirdPhrase is exactly TonicConclusivePhrase
```

Capítulo 6

Desarrollo del proyecto

En el siguiente capítulo expondremos la arquitectura general de nuestro proyecto, explicando independientemente el funcionamiento de cada uno de los módulos así como su relación con el resto. Expondremos el proceso de creación de los corpus de entrenamiento, así como el diseño e implementación del modelo de lenguaje basado en cadenas de Markov. Por último, mostraremos el desarrollo del sistema generador musical, en el que se han implementado las definiciones del clases y reglas establecidas en la ontología musical, con las que se controlarán los aspectos formales y armónicos de las obras generadas.

6.1 Arquitectura del proyecto

El objetivo principal de nuestro proyecto es la generación de melodías acompañadas emulando un determinado estilo musical, mediante la utilización de modelos de Markov que han sido generados mediante un proceso de minería de corpus. Como hemos visto en la sección del Estado de la Cuestión, el uso de esto modelos ha sido muy popular en las primeras décadas de la historia de inteligencia artificial, sin embargo los experimentos más relevantes que hemos analizado muestran una serie de deficiencias:

- Se limitan normalmente a la generación de melodías sencillas y cortas.

- Limitan a un pequeño número los acordes utilizados para la extracción de características armónicas.
- Limitan el tipo de figuraciones rítmicas posibles, utilizando ritmos sencillos, excluyendo grupos irregulares, silencios, etc.
- Muestran un comportamiento errático, con falta de direccionalidad, para órdenes de Markov bajos.
- Muestran *overfitting* o sobreentrenamiento cuando se utilizan órdenes de Markov altos, reproduciendo pasajes íntegros de las obras que se han utilizado para el aprendizaje.

6.1.1 Innovaciones aportadas

En nuestro proyecto introduciremos una serie de innovaciones para mejorar los resultados generativos del modelo que describiremos a continuación.

En primer lugar, hemos diseñado una ontología musical pesada a partir de un conocimiento experto, en la que se han definido algunas de las estructuras armónicas y rítmicas básicas de la música tonal, en forma de definiciones de clases de dominio como los grados musicales, las cadencias, las frases o las formas musicales, así como reglas entre las mismas. Dichas reglas se han implementado en el módulo compositor, de tal manera que las estructuras musicales armónicas y rítmicas generadas de manera estocástica mediante los modelos de Markov serán restringidas al cumplimiento de las reglas y estructuras ontológicas. La aplicación de estas reglas formuladas en la ontología corregirá la ausencia de direccionalidad encontrada típicamente en las producciones obtenidas por modelos markovianos de orden bajo; permitiéndonos de esta manera usar modelos de orden dos y tres, mediante su representación en bigramas y trigramas, respectivamente, sin caer en el problema del sobreentrenamiento obtenido por órdenes superiores, y modelando, de manera independiente, el plano armónico y el plano rítmico.

En segundo lugar, gracias a la gran capacidad de análisis musicológico del toolkit *music21* podremos obtener los símbolos del cifrado de cualquier acorde, incluyendo cuatríadas o quintíadas en cualquiera de sus inversiones, ampliando de esta manera el lexicón armónico y realizando una extracción de características armónicas más fidedigna: incluyéndose, por ejemplo, cifrados musicales que expresan breves modulaciones, dominantes secundarias, armonía alterada, etc. El análisis se realizará de forma independiente de la tonalidad: nuestro sistema detectará auto-

máticamente la tonalidad de la obra y obtendrá un cifrado romano independiente de la nota tónica, que expresa los grados de la escala, no las notas[1].

Con respecto a las características rítmicas, el toolkit nos permite trabajar, tanto con silencios como con notas, con cualquier duración que sea expresable en términos de la notación musical simbólica, sin restricción de ningún tipo.

Por último, nuestro sistema emulará la composición de obras enteras en un estilo determinado, permitiendo al compositor la elección del corpus (es decir, del estilo o compositor) utilizado, así como la duración de la obra en número de compases, el ritmo armónico (número de acordes por compás), la tonalidad de la misma, su estructura y el tempo musical. Las obras serán de tipo melodía acompañada y se escribirán dos pentagramas, guardándose en el disco duro bajo el formato estandarizado MusicXML, para su posterior análisis y reproducción en cualquier programa de edición musical.

6.1.2 Descripción de los distintos componentes

La arquitectura de nuestro sistema pone en evidencia la generalidad del enfoque empleado y constituye, por su reproducibilidad, una interesante aportación a la investigación, ya que hace patente su versatilidad, así como su posible utilidad para cualquier otro estilo musical o reglas estructurales y compositivas.

Nuestro sistema puede esquematizarse de la siguiente forma:

- El módulo de extracción de datos *en crudo* (*raw data*) desde las partituras presentes en los distintos corpus, codificadas en formato MusicXML.
- El módulo de análisis, que a partir de los datos en crudo, procesará los distintos símbolos y generará los bigramas y trigramas de cada estilo musical, para la armonía y el ritmo, además de aplicar técnicas procedentes del procesamiento del lenguaje natural como el suavizado o la eliminación de *stop words*.
- El módulo de composición, que generará música estocásticamente a partir de los modelos markovianos entrenados, sometiendo los resultados obtenidos a una serie de restricciones armónicas, rítmicas y formales definidas por las clases y reglas establecidas en la ontología musical.

[1] Las notas musicales (do, re, mi, etc...) dependen de la tonalidad, es decir, de la nota principal de la composición. Sin embargo, los grados tonales (I, II, III, etc.) constituyen una generalización de las mismas.

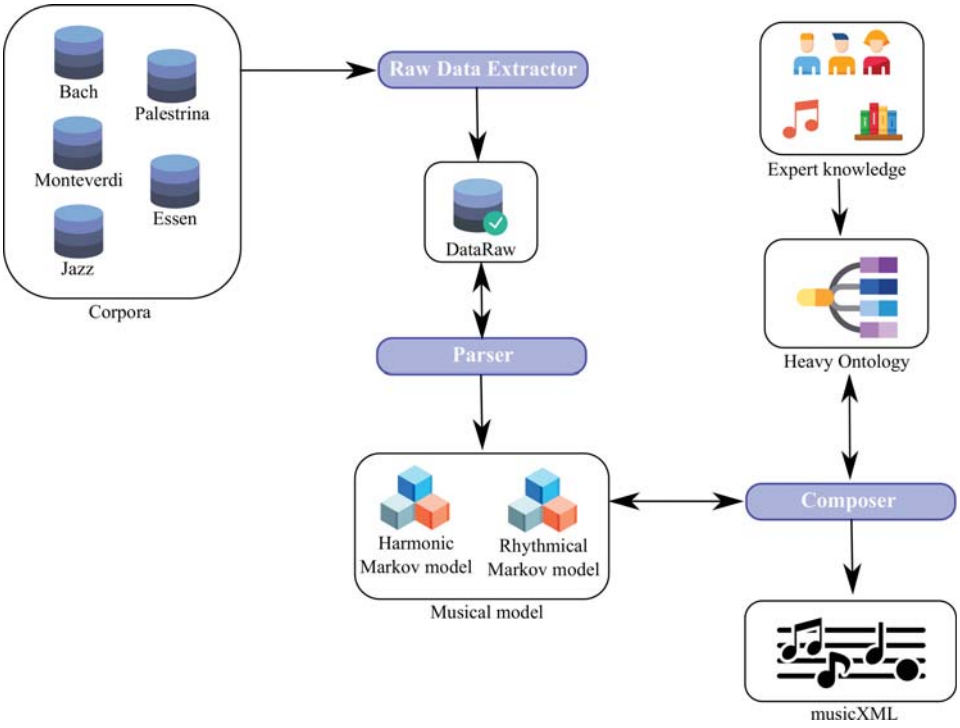


Figura 6.1: Arquitectura del proyecto.

6.2 Selección de los corpus de *training* y de *test*

El paquete de herramientas *music21* nos proporciona una gran variedad de obras compuestas por compositores de diversos periodos musicales, todas ellas en formato MusicXML a partir del cual podremos extraer las características que necesitemos. Encontramos 433 corales de J. S. Bach, 97 madrigales de Claudio Monteverdi, 1318 obras polifónicas de Giovanni Palestrina, así como la *Essen folk Collection* un recopilatorio de música folk que incluye miles de canciones de países de habla no inglesa, en particular de alemanas (Schaffrath, 1995). Además, en la página web `www.musescore.org` hemos recopilado un total de partituras de *standards* de *jazz*, también en formato MusicXML. Con estas fuentes, seleccionamos los corpus de los siguientes compositores, asumiendo el criterio expuesto en Martin y Jurafsky (2009) de asignar el 80 % de las obras al corpus de training y el 20 % restante al corpus de test.

- J. S. Bach: 346 corales para el corpus de training y 87 para el corpus de test.
- Claudio Monteverdi: 77 madrigales para el corpus de training y 20 para el corpus de test.
- Giovanni Palestrina: 1054 obras para el corpus de training y 264 para el corpus de test.
- Colección Essen folk: 4000 para el corpus de training y 1000 para el corpus de test.
- Estándares de jazz: 80 para el corpus de training y 20 para el corpus de test.

6.3 Creación del modelo de lenguaje musical

A continuación explicaremos los distintos procesos utilizados para el diseño del lenguaje musical, haciendo énfasis en la extracción de las características musicales armónicas y rítmicas, procesado de las secuencias de símbolos obtenidas y generación de bigramas y trigramas mediante el toolkit NLTK.

6.3.1 Extracción de características

Para la extracción de características desde las obras musicales, escritas frecuentemente a múltiples voces, ya sean de carácter homofónico o polifónico, utilizaremos el paquete de herramientas de music21, en el cual encontraremos un gran número de funcionalidades para extraer información desde la partitura.

Características armónicas

Un acorde se caracteriza por tener, al menos, tres notas musicales sonando simultáneamente. La disposición de estas notas, es decir, el orden en el que aparecen contando desde la nota más grave (bajo) hasta la más aguda, determina los que en términos de la armonía funcional se denomina *inversión*. Por tanto, podemos encontrar que un acorde, constituido por las mismas notas, aparezca en disposiciones diferentes y desempeñe por tanto un papel jerárquico diferente en el conjunto de total de acordes que se utilizan en el discurso musical. En la siguiente figura podemos ver las inversiones posibles del acorde de *Do* mayor, compuesto por las notas *Do*, *Mi*, y *Sol*

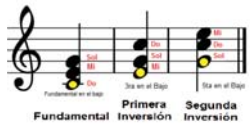


Figura 6.2: Ejemplo de las tres inversiones del acorde de *Do* mayor (*Inversión de un acorde*).

Distinguiremos entre la nota *fundamental* del acorde, que es la nota desde la que se genera el mismo mediante una sucesión de intervalos de tercera, y la nota del *bajo*, que es la nota que, en una determinada inversión del acorde, ocupa la posición más grave.

Pronto se pone de manifiesto que para caracterizar un acorde no lo es necesario encontrar cuáles son sus notas constitutivas (notas reales), sino también determinar el orden en el que éstas aparecen, contando desde la nota más grave. La disciplina de la armonía en la música tonal tradicional se encarga de sistematizar y establecer las relaciones entre la enorme casuística de acordes de tres, cuatro y cinco notas existente. Existen numerosas formas de caracterizar unívocamente mediante un símbolo el estado de un acorde, a esto se le denomina *cifrado*.

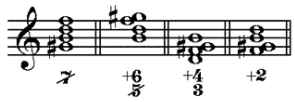


Figura 6.3: Ejemplo de las cuatro inversiones de un acorde de cuatro notas, así como de sus respectivos cifrados (Alvira, 2009).

El kit de herramientas music21 nos ofrece la funcionalidad *chordify*, mediante la cual podemos reducir a acodes cualquier tipo de música codificada en formato MusicXML, tal y como podemos ver ene l siguiente fragmento de código extraído de la documentación de music21:

```
from music21 import *
b = corpus.parse('bwv66.6')
bChords = b.chordify()
bChords.show()
```

El cifrado de acordes

Una vez reducida la partitura a una sucesión de acordes, con la funcionalidad *roman.romanNumeralFromChord* y conociendo la tonalidad (*key*) de la obra, podemos obtener el cifrado de cada acorde que lo identifica dentro del entramado de acordes funcionales de la obra.

```
for c in bChords.recurse().getElementsByClass('Chord'):
    rn = roman.romanNumeralFromChord(c, key.Key('A'))
    c.addLyric(str(rn.figure))
```

Si ejecutamos el código anterior en python, obtendremos la siguiente partitura:

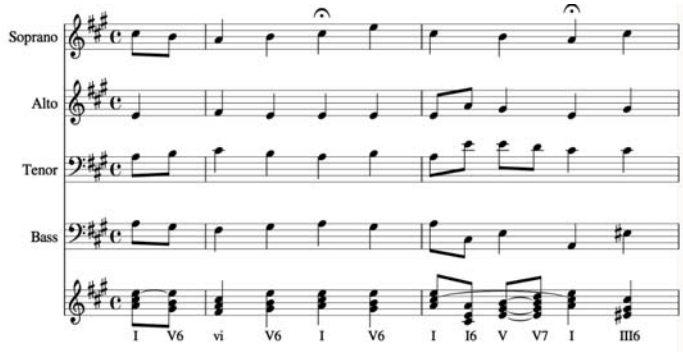


Figura 6.4: Ejemplo de las cuatro inversiones de un acorde de cuatro notas, así como de sus respectivos cifrados (Scott, 2019).

Encontrando que la secuencia de acordes empleados en la obra es la siguiente: *I, V6, vi, V6, I, V6, I, I6, V, V7, I, III6*. Mediante este proceso, podemos realizar un *parsing* de todas las obras de nuestros diferentes corpus y extraer el cifrado de cada acorde, lo que constituirá nuestra característica para la definición de la armonía.

Es remarcable mencionar que *music21* no sólo es capaz de reducir un acorde a su cifrado; también realiza el proceso contrario, es decir, dada una tonalidad, nos proporciona las notas y el orden de las mismas para cada uno de estos cifrados, con lo que tenemos resuelto el problema de la codificación de la armonía.

Características rítmicas

La extracción de características rítmicas se realizará mediante la propiedad *quarterLength* implícita en la clase *duration.Duration* del toolkit *music21*. Dicha propiedad devuelve un valor numérico que expresa de manera unívoca[2] el número de negras contenidas en la duración de la nota. Recorriendo todas las notas de cada voz melódica acumulamos los símbolos, en este caso numéricos, de las diferentes duraciones de cada nota. Añadiremos a este símbolo una «N» para el caso de encontrar una nota y una «R» para el caso del silencio, por lo que el análisis de cualquier combinación rítmica, por compleja o irregular que sea, está asegurada. En el siguiente ejemplo podemos ver el valor de quarterLength de una blanca con dos puntillos:

```
d = duration.Duration(type='half', dots=2)
d.quarterLength
>> 3.5
```

En caso de encontrar una nota con esta duración, almacenaríamos su duración con el símbolo «N3.5», en caso de ser un silencio, con el símbolo «R3.5».

6.3.2 Parsing de las obras

Como el proceso de *chordify* y la posterior extracción de su cifrado con mediante *roman* es bastante lento, realizaremos un preprocesado de todas las obras del corpus, extrayendo el cifrado de todos los acordes de las mismas y almacenándolos estos datos en «crudo» en un fichero de texto para cada compositor. Para ello utilizaremos las función que hemos escrito denominada *getRawData*, y posteriormente lo almacenaremos en un archivo de texto.

El proceso *getRawData* realizará este *parsing* distinguiendo las obras que estén en tonalidad mayor de las que estén en tonalidad menor, proporcionando de esta forma dos listas de símbolos distintas. Esta distinción constituye un punto fundamental en nuestra investigación, ya que consideramos que el tipo de acordes que se utilizan, así como las relaciones existentes entre los mismos, cambian de manera sustancial en los modos mayor y menor propios de la música tonal.

```
from music21 import *
```

[2]Cada figura rítmica se corresponde con una y sólo una duración en quarterLenght. Para más información sobre esta propiedad véase `http://web.mit.edu/music21/doc/moduleReference/moduleDuration.html`.

```
obras = corpus.getComposer('bach', 'xml')
dataRaw = getRawData(obras)
writeText(dataRaw[0] , "dataRawBachMajor.txt")
writeText(dataRaw[1] , "dataRawBachMinor.txt")
```

La función *getRawData* analizará una por una cada una de las obras existentes en cada corpus. Detectará automáticamente la tonalidad de cada obra, gracias a la funcionalidad incluida en *music21 getElementsByClass(key.KeySignature)* y a continuación extraeremos el cifrado de cada acorde.

```
bChords = b.chordify()
chords = bChords.recurse().getElementsByClass('Chord')
ks=b.flat.getElementsByClass(key.KeySignature)[0]
for c in chords:
    rn = roman.romanNumeralFromChord(c, ks)
    if(ks.mode=="major"):
        majorMode = majorMode + " " + str(rn.figure)
    elif(ks.mode=="minor"):
        minorMode = minorMode + " " + str(rn.figure)
```

Repetiremos este proceso con todos los corpus seleccionados anteriormente, extrayendo la información armónica existente en cada corpus codificada en forma de secuencias de símbolos, que serán almacenadas en dos archivos para cada corpus, uno para el modo mayor y otro para el modo menor.

6.3.3 Tokenización

Una vez extraídos los acordes pertenecientes a cada corpus, tanto para el modo mayor como el menor, procederemos a la *tokenización* del texto. Se realizará identificando cada palabra separando los símbolos por el carácter de espacio. Dejaremos la distinción entre mayúsculas y minúsculas, ya que el cifrado requiere de esta discriminación, así como los símbolos repetidos, ya que pueden formar parte del discurso armónico. Asumiremos que el vocabulario es cerrado, es decir, no hay más símbolos (acordes) que aquellos que encontremos en el corpus de training.

6.3.4 *Eliminación de las* stop words

En el proceso de *parsing*, a menudo music21 aglutina en acordes algunas notas que no corresponden realmente al mismo, como notas de paso, bordaduras, floreos, etc. Realizaremos una lista de estos símbolos no convenientes, irrelevantes desde el punto de vista de la armonía funcional y los trataremos como *stop words* que tendrán que ser eliminadas del conjunto de palabras.

6.3.5 *Frecuencia de símbolos*

Gracias a las herramientas proporcionadas por *NLTK* podemos visualizar gráficamente los acordes más utilizados en cada uno de los corpus, a continuación mostraremos los resultados armónicos y rítmicos obtenidos para los distintos corpus de cada compositor.

Corpus de J. S. Bach

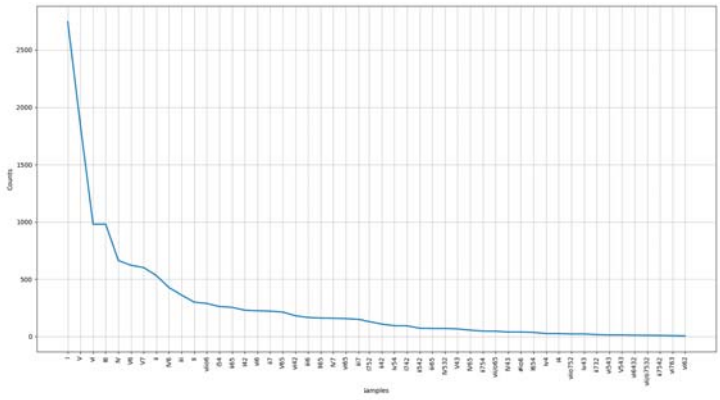


Figura 6.5: Frecuencia de palabras armónicas para el modo mayor en el corpus J. S. Bach.

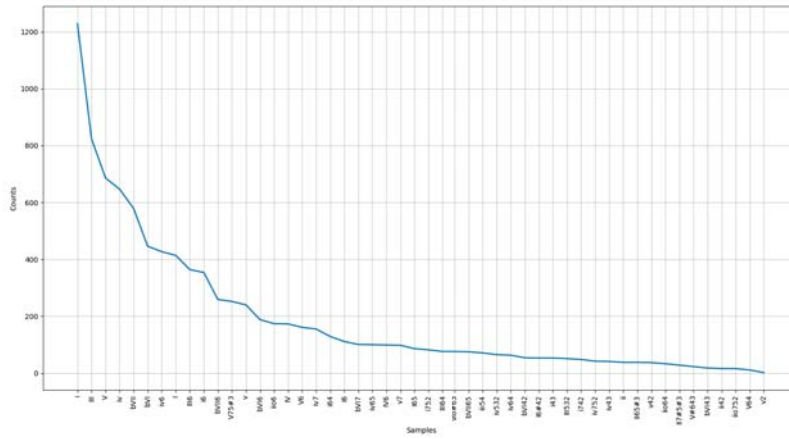


Figura 6.6: Frecuencia de palabras armónicas para el modo menor en el corpus J. S. Bach.

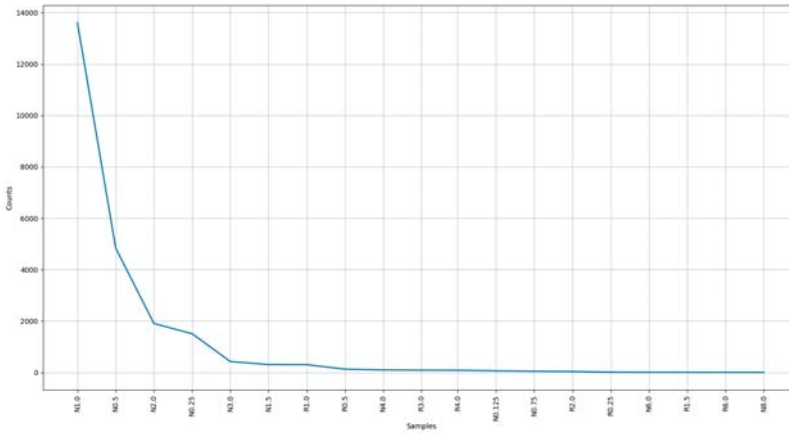


Figura 6.7: Frecuencia de palabras rítmicas en el corpus J. S. Bach.

Corpus de Monteverdi

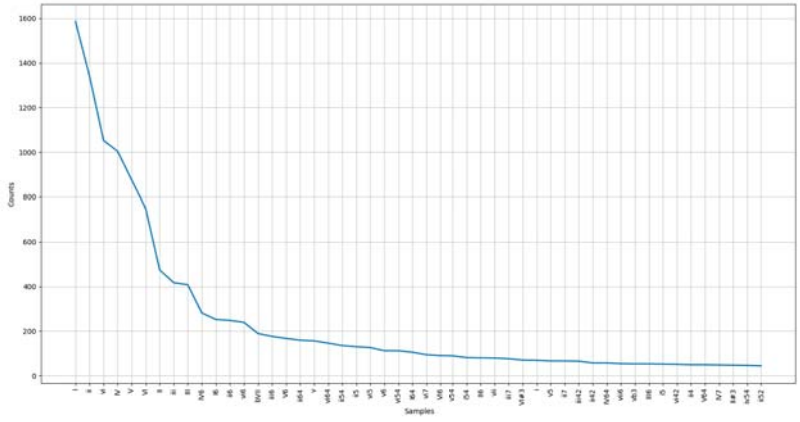


Figura 6.8: Frecuencia de palabras armónicas para el modo mayor en el corpus Monteverdi.

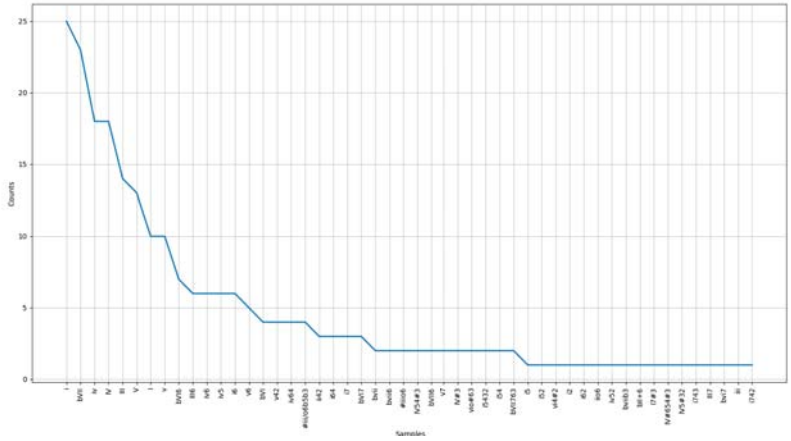


Figura 6.9: Frecuencia de palabras armónicas para el modo menor en el corpus Monteverdi.

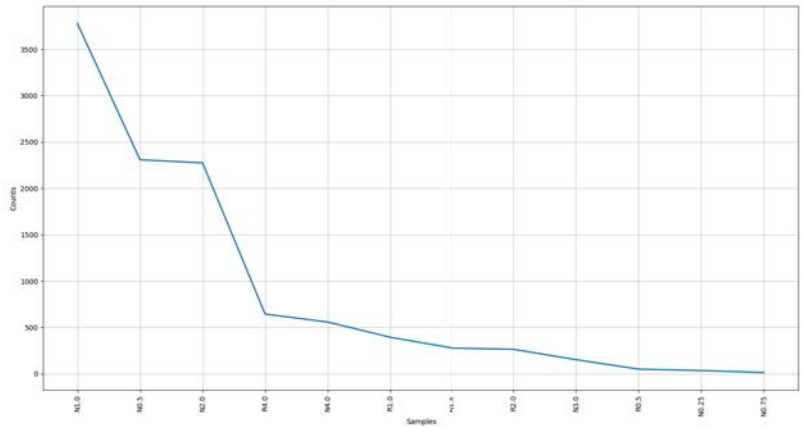


Figura 6.10: Frecuencia de palabras rítmicas en el corpus Monteverdi.

Corpus de Palestrina

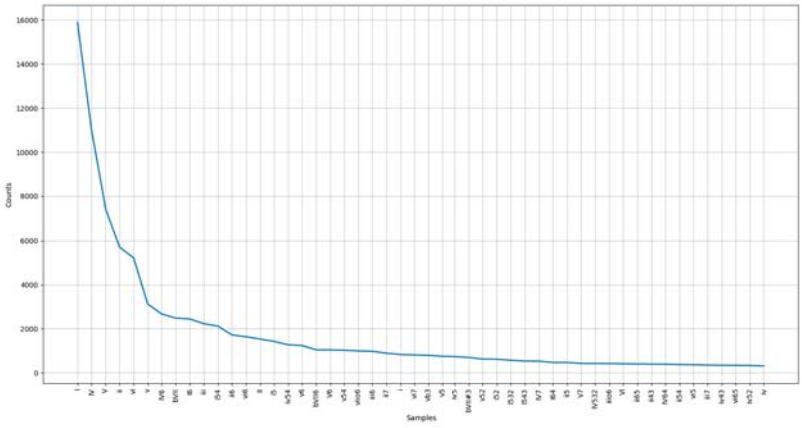


Figura 6.11: Frecuencia de palabras armónicas para el modo mayor en el corpus Palestrina.

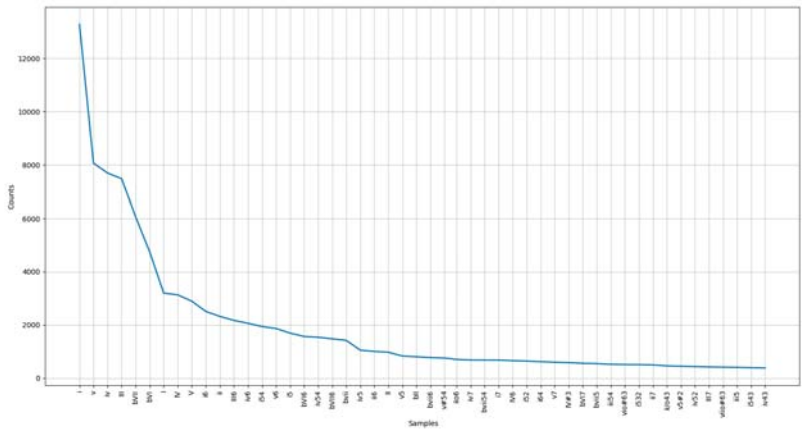


Figura 6.12: Frecuencia de palabras armónicas para el modo menor en el corpus Palestrina.

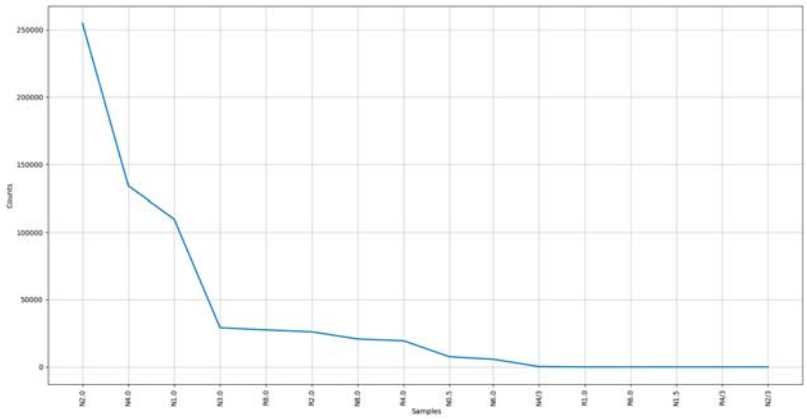


Figura 6.13: Frecuencia de palabras rítmicas en el corpus Palestrina.

Corpus de Essen Folksongs

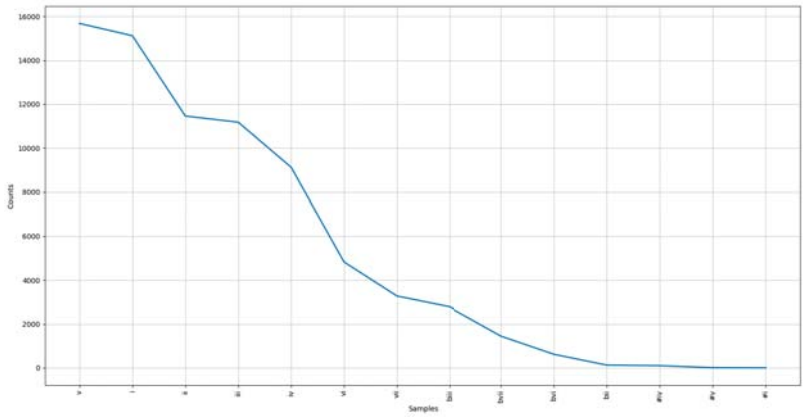


Figura 6.14: Frecuencia de palabras armónicas para el modo mayor en el corpus Essen.

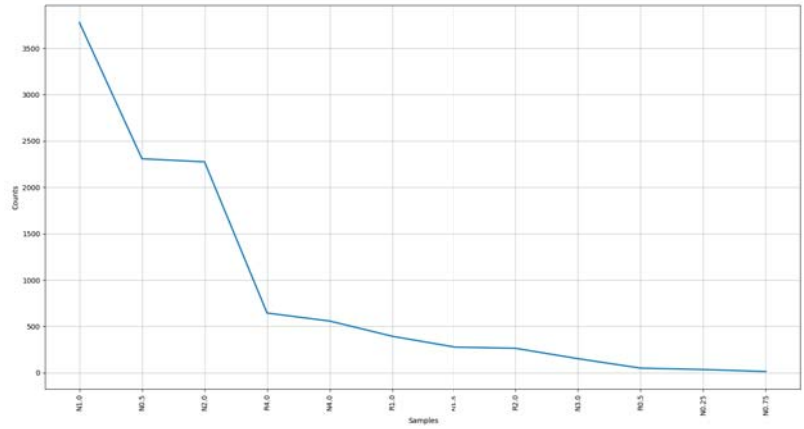


Figura 6.15: Frecuencia de palabras rítmicas en el corpus Essen.

Corpus de Jazz

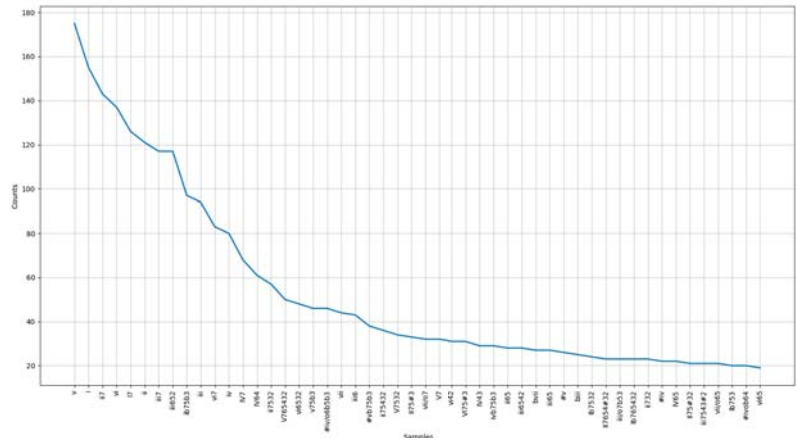


Figura 6.16: Frecuencia de palabras armónicas para el modo mayor en el corpus Jazz.

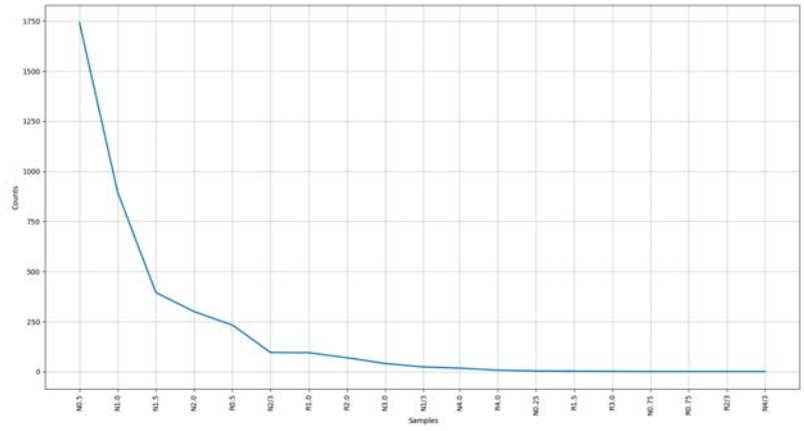


Figura 6.17: Frecuencia de palabras rítmicas en el corpus Jazz.

6.3.6 Generación del bigrama y trigrama

La clase ngram contiene métodos para la generación de los bigramas y los trigramas utilizando herramientas proporcionadas por el toolkit NLTK. Las funciones *getBigram* y *getTrigram* realizarán el conteo de bigramas y trigramas existentes en el total de tokens.

```
def getBigram(self, tokens):
    bigrm = nltk.bigrams(tokens)
    bigrm_count = {}
    for grm in bigrm:
        if grm not in bigrm_count:
            bigrm_count[grm] = 1
        else:
            bigrm_count[grm] += 1
    return bigrm_count


def getTrigram(self, tokens):
    trigrm = nltk.trigrams(tokens)
    trigrm_count = {}
    for grm in trigrm:
        if grm not in trigrm_count:
            trigrm_count[grm] = 1
        else:
            trigrm_count[grm] += 1
    return trigrm_count
```

Los bigramas y trigramas son matrices de tipo *sparse*, es decir, la mayoría de sus elementos valen cero. Las representaremos mediante diccionarios que acumularán únicamente las duplas o triplas de valores de tokens para los cuales se ha encontrado al menos un conteo en el corpus correspondiente.

6.3.7 Smoothing de los bigramas y trigramas

Se aplicará el *smoothing* aditivo (Laplace) a los bigramas y trigramas generados anteriormente. Se almacenará en un diccionario la probabilidad calculada mediante este suavizado y en otro diccionario (de orden una dimensión menor) la probabilidad en el caso en el que el conteo ha sido cero (para evitar el problema anteriormente mencionado del tamaño de la matriz *sparse*). Recordemos que la fórmula para el smoothing aditivo, para un conjunto de observaciones

$X = \langle x_1, x_2, \cdots, x_d \rangle$ es:

$$P_i = \frac{x_i + \alpha}{N + \alpha d} \quad (i = 1, \cdots, d)$$

Las funciones utilizadas para esta labor, cuyo código puede ser leído en la sección de apéndices, se encuentran en la clase *smoothing* y son las siguientes:

```
calculateBigramProbabilityLaplace(self, unigram, bigram, alfa)

smoothingLaplaceBigramZERO(self, unigram, bigram, alfa)

calculateTrigramProbabilityLaplace(self, unigram, trigram, alfa)

smoothingLaplaceTrigramZERO(self, unigram, trigram, alfa)
```

Los diccionarios de bigramas y trigramas generados serán almacenados en ficheros binarios con extensión *.pkl* mediante el módulo *pickle*, para su posterior recuperación en la fase de generación de material musical.

```
def save_obj(self, obj, name):
    with open('obj/'+ name + '.pkl', 'wb') as f:
        pickle.dump(obj, f, pickle.HIGHEST_PROTOCOL)

def load_obj(self, name):
    with open('obj/' + name + '.pkl', 'rb') as f:
        return pickle.load(f)
```

6.4 Implementación del sistema compositivo

A continuación describiremos el proceso de diseño del sistema compositivo, mediante el cual se generan melodías acompañadas con acordes en el estilo de un determinado compositor.

6.4.1 Determinación de la estructura formal

La clase encargada de realizar el proceso compositivo es *composer*. Utilizará el modelo de lenguaje musical basado en los bigramas y trigramas, tanto armónicos como rítmicos, generados anteriormente. El usuario selecciona el *compositor* y la tonalidad en la que quiere hacer la composición, así como la forma, duración de la misma (forma binaria o ternaria) y ritmo armónico. El sistema generará las frases melódicas necesarias para completar la forma musical requerida, imponiendo las restricciones armónicas y cadenciales establecidas por la ontología musical.

Por ejemplo, para una forma binaria de 16 compases, el sistema generará dos semifrases. La primera semifrase, de ocho compases, será de tipo tónica, es decir, irá del primer grado tonal al quinto grado tonal. La segunda frase, también de ocho compases, será de tipo dominante, es decir, comenzará en el quinto grado tonal y finalizará en el primer grado tonal mediante una cadencia perfecta. El proceso sería el siguiente:

- Generación de una secuencia armónica para los ocho compases de la primera semifrase, comenzando con I y finalizando con una cadencia permitida en la frase tónica.
- Generación de una secuencia rítmica de ocho compases.
- Creación de la melodía de la primera semifrase: asignación de alturas a cada uno de los elementos rítmicos, según el acorde que le corresponda.
- Generación de una segunda secuencia armónica para los ocho compases de la segunda semifrase de dominante, comenzando con V y finalizando con I, con cadencia perfecta.
- Generación de una secuencia rítmica de 8 compases.
- Creación de la melodía de la segunda semifrase, asignando las alturas a cada uno de los elementos rítmicos, según el acorde que le corresponda.
- Generación de la partitura, uniendo la melodía y los acordes generados para la primera semifrase y segunda semifrase.

Este proceso se realizaría de forma análoga para formas musicales más complejas como por ejemplo la forma ternaria, en la que hay tres semifrases (tónica, dominante y tónica conclusiva), siguiendo las reglas definidas por la ontología.

6.4.2 Generación de una secuencia armónica

La función encargada de la generación de las secuencias armónicas se encuentra en la clase *harmony*

```
getHarmony_from_Trigram(trigram, bigram, tonality, "I", numberOfChords, duration
```

Debemos proporcionarle el bigrama y el trigrama del modelo de lenguaje armónico, la tonalidad así como el primer símbolo del lexicón, que le será proporcionado según las reglas de la ontología para cada tipo de semifrase. La función utilizará las probabilidades contenidas en el bigrama para determinar aleatoriamente, mediante una distribución de probabilidad uniforme, cuál es el siguiente símbolo generado. Se le proporcionará también el número de acordes y la duración de cada uno de ellos, para determinar de esta manera la duración total de la secuencia armónica generada. La función encargada de obtener un símbolo aleatorio según las probabilidades del bigrama o trigrama es *getRandomTuple()*, presente en la clase *ngram*.

6.4.3 Generación de una secuencia rítmica

La función que generará una secuencia rítmica mediante los bigramas y trigramas del modelo de lenguaje rítmico se encuentra en la clase *rhythm* y es la siguiente:

```
getRhythm_from_Trigram(trigram_R, "N2.0", "N1.0", ritmoLenght)
```

A esta función tenemos que proporcionarle el trigrama rítmico, los dos primeros elementos rítmicos así como la duración total de la secuencia generada. En la generación de secuencias rítmicas se tendrá que cumplir una serie de reglas, necesarias para la coherencia del ritmo generado:

- Los tresillos de corchea sólo podrán aparecer en la parte fuerte del tiempo del compás.
- Los tresillos de negra sólo podrán aparecer en la parte fuerte del compás.
- No se puede exceder una duración acumulada de silencios superior a dos negras.

Además, se permite seleccionar entre la introducción manual de los dos primero elementos rítmicos, o la selección aleatoria de los mismos, obteniendo con este segundo método una mayor variedad en los resultados.

6.4.4 Asignación de alturas a la secuencia rítmica

Una vez generadas las secuencias armónicas y rítmicas, ambas de igual duración, se asignará la nota de cada elemento rítmico según las notas del acorde que le corresponda. Se elegirá la nota aleatoriamente entre todas las notas del acorde, atendiendo a las siguientes restricciones:

- No se puede repetir nota, es decir, la nueva nota seleccionada no puede ser igual a la nota inmediatamente anterior.
- La distancia en semitonos de la nota seleccionada con la nota inmediatamente anterior ha de ser la mínima posible.
- Se tiene que respetar la tesitura de la voz de la soprano, que abarca desde un Do_4 hasta un La_5.

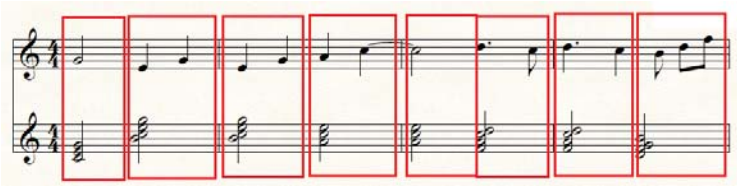


Figura 6.18: Ejemplo de asignación de notas a la secuencia rítmica siguiendo con las reglas anteriormente explicadas.

6.4.5 Escritura de la partitura final

Una vez generadas las dos semifrases melódicas, acompañadas de sus dos respectivas semifrases armónicas, procederemos a escribir la partitura generada en formato MusicXML para que pueda ser escuchada desde un software editor de partituras cualquiera. Realizaremos este proceso gracias a las funcionalidades del toolkit *music21*, implementadas en la función *generateScore* de la clase *composer*:

```
def generateScore(self, chords, melody, tonality):
    s1 = stream.Stream()
```

```
p1 = stream.Part()
k1 = tonality
p1.append(k1)
if(melody is not None):
    for m in melody:
        p1.append(m)
    s1.append(p1)
p2 = stream.Part()
p2.append(k1)
for c in chords:
    p2.append(c)
s1.append(p2)
return s1
```

De esta manera, el sistema es capaz de generar automáticamente cientos de composiciones musicales completas, coherentes desde el punto de vista formal con las definiciones y restricciones armónicas y estructurales expuestas en la ontología.

Capítulo 7

Resultados

En el siguiente capítulo expondremos algunas composiciones musicales generadas por nuestro sistema, además de mostrar los resultados obtenidos por el mismo en la fase de evaluación, comparando su rendimiento con otras investigaciones similares.

7.1 Resultados obtenidos

Nuestro sistema es capaz de generar composiciones musicales emulando un total de cinco estilos diferentes: J. S. Bach, Monteverdi, Palestrina, Jazz y canciones folk (corpus de *Essen folksong*). Permite al usuario seleccionar uno de estos cinco estilos y configurar además la estructura formal de la composición, la duración en compases de la obra generada, la tonalidad de la pieza, el ritmo armónico (un acorde nuevo cada compás, dos acorde cada compás, etc.) y el tempo. El sistema utiliza la implementación de la ontología diseñada para construir, mediante reglas armónicas y estructurales, las frases y subfrases que constituyen cada forma musical. Las obras resultantes son del tipo melodía acompañada, se transcriben en dos pentagramas distintos, uno para melodía y otro para acordes, y se exportan a archivo en formato estandarizado MusicXML.

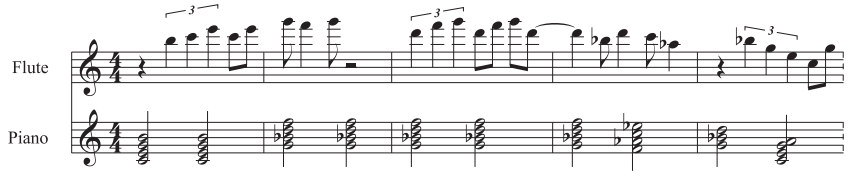


Figura 7.1: Ejemplo de un fragmento musical generado en el estilo Jazz.
`https://soundcloud.com/brian-martinez-481263365/harmonic-voice-jazz`

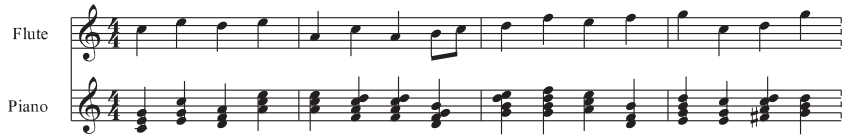


Figura 7.2: Ejemplo de un fragmento musical generado en el estilo J. S. Bach.
`https://soundcloud.com/brian-martinez-481263365/harmonic-voice-bach`

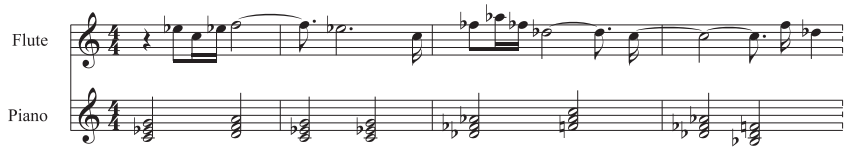


Figura 7.3: Ejemplo de un fragmento musical generado en el estilo folk de Essen.
`https://soundcloud.com/brian-martinez-481263365/harmonic-voice-folk`

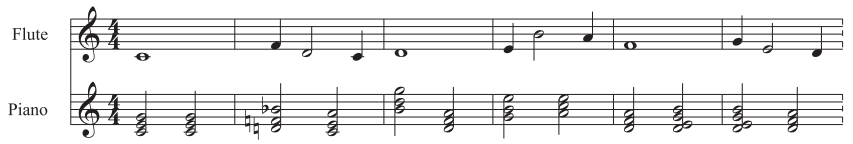


Figura 7.4: Ejemplo de un fragmento musical generado en el estilo de Monteverdi.
`https://soundcloud.com/brian-martinez-481263365/harmonic-voice-monteverdi`

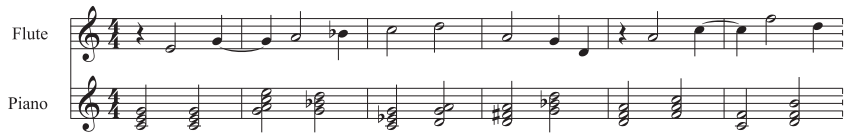


Figura 7.5: Ejemplo de un fragmento musical generado en el estilo de Palestrina.
`https://soundcloud.com/brian-martinez-481263365/harmonic-voice-palestrina`

En la siguiente url se pueden escuchar otras composiciones generadas:
`https://soundcloud.com/brian-martinez-481263365/sets/harmonic-voice`

7.2 Proceso de evaluación

La evaluación de estilos puede considerarse un proceso relativo a la inteligencia artificial, por lo que podemos aplicar para su evaluación, al menos parcialmente, un *test de Turing*.

7.2.1 Evaluaciones realizadas en otros estudios previos

Para ello realizaremos un diseño experimental similar a los realizados por Schulze y Van Der Merwe (2010) y Roig y col. (2014), con la finalidad de establecer una metodología que permita la comparación de nuestros resultados obtenidos con los de otros estudios.

En el primer artículo se realizan dos encuestas para cada estilo musical generado. La primera encuesta consiste en un test de Turing realizado *online* para que el usuario detecte si una determinada composición ha sido compuesta por un humano o por la máquina. La segunda encuesta consiste en evaluar tres composiciones (una humana y dos generadas por ordenador) por orden de preferencia.

En el segundo artículo se realiza una encuesta abierta sobre distintos conjuntos de tres melodías distintas (una humana y dos generadas por ordenador), de forma que cada participante evalúa un conjunto distinto. En cada conjunto, las melodías pertenecen a un mismo estilo. A cada participante se pregunta que detecte si cada composición es de origen humano o no, y se le pide que evalúe cada composición mediante una escala Likert valorada desde el 1 al 5.

7.2.2 Diseño del proceso de evaluación

Continuando con las evaluaciones de los anteriores artículos, prepararemos un diseño experimental similar para evaluar los resultados de nuestro sistema. Realizaremos tres encuestas con tres finalidades distintas: en la primera evaluaremos la identificación del estilo, en la segunda identificaremos la diferenciación entre composición humana o artificial (test de Turing), en la tercera evaluaremos la preferencia de la humana frente la artificial (test de preferencia).

La primera encuesta consistirá en preguntar un grupo de expertos por la identificación del estilo de una determinada obra. Los posibles estilos son J. S. Bach, Palestrina, Monteverdi, Jazz o música folk. Se utilizarán, para cada uno de los estilos, diez obras compuestas por un humano (procedentes del corpus de test) y diez obras compuestas por computador, mediante nuestro sistema. Los expertos serán preguntados aleatoriamente por la identificación del estilo de las 100 obras

totales que formará nuestro corpus de evaluación (50 compuestas por humano y 50 por ordenador). Anotaremos con un 1 cuando el experto haya acertado, y un 0 cuando se haya equivocado, tanto para obras de origen humano como computacional.

La segunda encuesta determinará el grado de diferenciación de las obras generadas por computador respecto a las obras generadas por humano. Para ello preguntaremos nuevamente al grupo de expertos, de manera aleatoria, si la obra ha sido compuesta por un humano o por ordenador. Anotaremos con un 1 cuando el experto haya acertado, y con un 0 cuando se haya equivocado. Repetiremos este procedimiento para cada estilo musical.

Por último, evaluaremos las preferencias musicales del grupo de expertos. Para ello seleccionaremos un conjunto de tres obras por cada estilo. Dos obras serán compuestas por computador mediante nuestro sistema, y una obra será compuesta por humano, extraída aleatoriamente del corpus de test. Marcaremos con un 1 la obra preferida por cada experto, y con un 0 las otras dos.

7.2.3 Grupo de expertos

Para realizar esta evaluación será necesario seleccionar un grupo de expertos, que en nuestro caso se compondrá por 43 alumnos del Conservatorio Superior de Música de Murcia, procedentes de diversas especialidades (instrumentistas, composición, dirección de orquesta y musicología), todos ellos muy bien entrenados en teoría musical, historia de la música y educación auditiva.

El proceso de muestreo para la selección del grupo experimental no es aleatorio, sino de conveniencia, ya que aprovechamos los individuos que tenemos más cercanos para realizar las encuestas. Las limitaciones de nuestra evaluación incluyen la no existencia de un grupo de control, formado por individuos seleccionados aleatoriamente por ejemplo mediante una encuesta pública online.

7.2.4 Restricciones de nuestras composiciones

Las obras generadas por computador se han adaptado a unos requisitos musicales de carácter técnico para su correcta comparación con las obras del corpus de test relativas a cada estilo, y para la correcta comparación de los resultados obtenidos con los resultados de otros experimentos. Dichas restricciones son las siguientes:

- El compás se mantiene constante a lo largo de toda la composición y es 4 por 4.

- La tonalidad se ha restringido a Do Mayor.
- El número de tiempos máximo de silencio permitidos es dos.
- Cada fraseo tiene que concluir con una cadencia (tal y como está definido en la Ontología musical que se ha implementado).
- En el caso de las obras generadas en el estilo de los corales J. S. Bach, la obra tiene una forma binaria de 8 compases de duración, comenzando con una frase tónica de cuatro compases y continuando con una segunda frase dominante también de cuatro compases, finalizando con cadencia perfecta (V-I).
- En el caso de las obras generadas en estilo Jazz, consisten en una forma ternaria A-B-A de 24 compases, formada por una frase tónica, frase dominante y frase tónica conclusiva.

  Además, para la correcta realización de la comparativa entre obras compuestas por humano y por nuestro sistema se han transportado las 50 obras de origen humano seleccionadas dentro corpus de evaluación a la tonalidad de Do y se han reducido sus acordes a su disposición básica, tal y como se genera en nuestro sistema.

7.3 Resultados de la evaluación

A continuación expondremos los resultados obtenidos para las encuestas realizadas sobre el grupo de expertos para cada uno de los diferentes estilos musicales.

7.3.1 Identificación del estilo

Analizando los datos obtenidos en los que se pregunta al grupo de expertos por el estilo de cada una de las obras del corpus de evaluación, obtenemos:

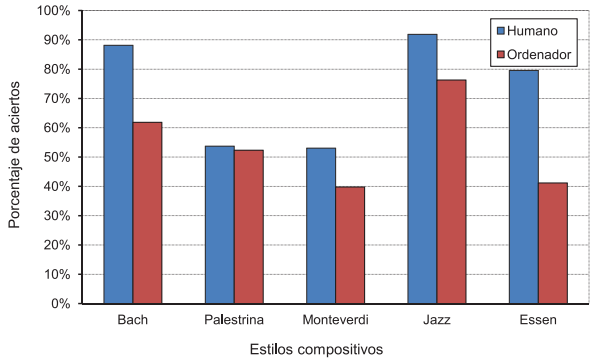


Figura 7.6: Porcentaje de aciertos para la identificación del estilo.

7.3.2 Identificación del compositor humano

Analizando los datos obtenidos en los que se pregunta al grupo de expertos, para todo el corpus de evaluación, si el compositor es un humano, obtenemos:

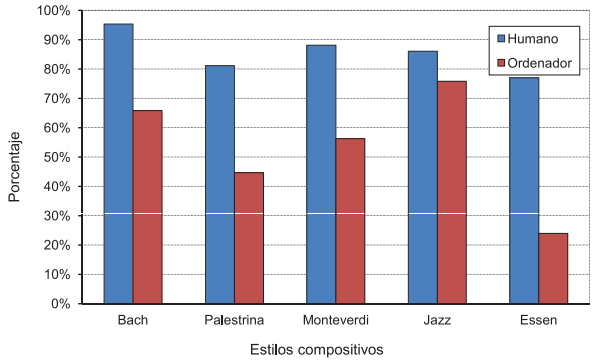


Figura 7.7: Porcentaje de aciertos para la identificación del compositor humano.

7.3.3 Preferencia de composición específica

Después de analizar los datos obtenidos para la encuesta en la que se pregunta al grupo de expertos, sobre un total de tres obras diferentes (dos de las cuales son generadas por ordenador y una es generada por humano) y para cada estilo, por la obra que le resulta preferida, obtenemos los siguientes resultados:

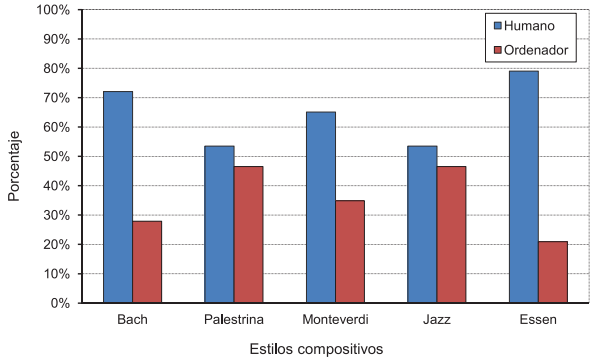


Figura 7.8: Comparativas de preferencia entre las obras compuestas por humano y por nuestro sistema para la cada estilo musical.

7.4 Comparación de resultados

En la siguiente tabla podemos ver los resultados obtenidos para la identificación de obras compuestas por nuestro sistema, comparándolos con los obtenidos por otros estudios. *Identificación incorrecta* significa el porcentaje de encuestados que fallaron a la hora de identificar como artificial la obra compuesta por nuestro sistema, mientras que *Identificación correcta* muestra el porcentaje de encuestados que descubrieron que efectivamente la obra había sido generada por nuestro sistema. El número entre paréntesis muestra en número total de encuestados. La siguiente tabla muestra una comparativa entre nuestros resultados, los resultados de Roig y col. (2014) y Schulze y Van Der Merwe (2010)

Tabla 7.1: Evaluación del test de Turing y comparativa con otros experimentos

Estilo	Identificación incorrecta	Identificación correcta
Bach	66 % (28)	34 % (15)
Palestrina	45 % (19)	55 % (24)
Monteverdi	57 % (25)	43 % (18)
Jazz	76 % (33)	24 % (10)
Essen	24 % (10)	76 % (33)
Pop (Roig y col., 2014)	71 % (63)	29 % (25)
Academic (Roig y col., 2014)	70 % (62)	30 % (26)
Flamenco (Roig y col., 2014)	58 % (51)	42 % (37)
Burger (Schulze y Van Der Merwe, 2010)	44 % (54)	56 % (70)
Melted (Schulze y Van Der Merwe, 2010)	34 % (47)	66 % (92)

A continuación mostraremos la evaluación con respecto a la preferencia de la obra humana frente la obra compuesta por el ordenador.

Tabla 7.2: Evaluación del test de preferencia y comparativa con otros experimentos

Estilo	Preferencia humano	Preferencia computador
Bach	72 % (31)	28 % (12)
Palestrina	53 % (23)	47 % (20)
Monteverdi	65 % (28)	35 % (15)
Jazz	53 % (23)	47 % (20)
Essen	79 % (34)	21 % (9)
Burger (Schulze y Van Der Merwe, 2010)	44 % (55)	56 % (70)
Melted (Schulze y Van Der Merwe, 2010)	38 % (53)	66 % (86)

Capítulo 8

Conclusiones

8.1 Conclusiones

En la presente investigación hemos realizado un sistema capaz de modelizar un lenguaje musical basado en un modelo de Markov sobre el que se implementan reglas armónicas y estructurales preestablecidas por el diseño de una ontología musical pesada. El sistema se muestra capaz de componer obras completas, coherentes desde el punto de vista de la estructura formal y armónica, mediante la combinación de la generación independiente de ritmos y armonía, calculados estocásticamente por los bigramas y trigramas, previamente entrenados en distintos corpus musicales pertenecientes a distintos estilos musicales.

En nuestro proyecto hemos introducido una serie de innovaciones para mejorar los resultados generativos del modelo con respecto a investigaciones anteriores:

En primer lugar, se ha diseñado una ontología musical pesada a partir de la elicitación de conocimiento experto, en la que se han definido estructuras y reglas tanto armónicas como rítmicas propias de la música tonal (grados musicales, las cadencias, las frases o las formas musicales). Dichas reglas han sido implementadas de tal manera que las estructuras musicales armónicas y rítmicas generadas de manera estocástica mediante los modelos de Markov serán restringidas al cumplimiento de las reglas y estructuras definidas en la ontología, corrigiendo la

ausencia de direccionalidad encontrada típicamente en las producciones obtenidas por modelos markovianos de orden bajo; permitiéndonos evitar el problema del sobreentrenamiento obtenido por órdenes superiores.

En segundo lugar, gracias a la utilización de nuevos frameworks para el análisis musicológico hemos incluido la totalidad de los símbolos relativos al cifrado de cualquier acorde, incluyendo cuatríadas o quintíadas, así como sus respectivas inversiones, utilizando de esta manera un amplio lexicón armónico. Esto nos permite incluir breves modulaciones, dominantes secundarias o armonía alterada, por ejemplo. El análisis se realizará de forma independiente de la tonalidad ya que nuestro sistema detecta automáticamente la tonalidad de la obra y obtiene un cifrado de carácter general independiente de la misma.

Con respecto a las características rítmicas, hemos conseguido incorporar tanto los silencios como las notas, para cualquier duración que sea expresable en términos de la notación musical simbólica, sin restricción de ningún tipo, incluso grupos irregulares como tresillos, quintillos, etc.

Por último, nuestro sistema es capaz de emular la composición de obras enteras en un estilo determinado, permitiendo elegir al compositor la el corpus a utilizar, así como el número de compases de la obra, su ritmo armónico, la tonalidad, la estructura y el tempo. Las obras producidas son de tipo melodía acompañada y se transcriben en dos pentagramas, almacenándose en el disco duro con el formato estandarizado MusicXML, que permite fácilmente su posterior análisis y reproducción en cualquier programa de edición musical.

8.1.1 Con respecto a la evaluación de resultados

En base al análisis de la evaluación a la que hemos sometido nuestros resultados podemos afirmar que el sistema que hemos diseñado en la presente investigación presenta unos buenos rendimientos, tanto en el test de Turing como en el test de preferencia, similares a los de otros experimentos. Sin embargo esta afirmación es demasiado general y conviene detallarla con mayor profundidad.

Identificación de estilo

1. Los resultados obtenidos en el primer test, la identificación de estilo, ponen de manifiesto que los estilos más fácilmente identificables son Bach, Jazz y Folk. Sin embargo, los estilos que generan más confusión en el grupo de expertos son Monteverdi y Palestrina. Esto ha sucedido tanto para la identificación de obras compuestas por un humano como para las obras generadas por nuestro sistema.

2. Con referencia a las obras compuestas por nuestro sistema, obtienen porcentajes de acierto en la identificación de su estilo inferiores a las obras compuestas por humano. Destaca en este punto el estilo Jazz, que con un 76 % de aciertos es el estilo que mejor emula nuestro sistema. Le siguen Bach, con un 62 % de aciertos, Palestrina (52 %) y por último Essen (41 %) y Monteverdi (40 %), ambos con resultados modestos.

3. Los autores Roig y col. (2014) exponen en su artículo unos resultados para la estimación del nivel de semejanza de 3,6 sobre 5,0 (72 %)para el estilo pop, 3,2 sobre 5,0 (64 %) para el estilo académico y 3,93 sobre 5,0 (79 %) para el estilo flamenco, similares a los obtenidos por nuestro sistema.

Test de Turing

Los resultados para la evaluación del test de Turing ofrecen unos rendimientos dispares, encontrando que para el estilo de Jazz se alcanza un muy buen resultado, con un 76 % de expertos equivocados (es decir, han valorado compositor humano una obra realizada por nuestro sistema) y un 66 % de expertos equivocados en el caso del estilo de Bach. Analizando estos resultados con mayor detalle:

1. Los resultados para el test de Turing del estilo Jazz alcanza un 76 % de expertos equivocados, superando los mejores resultados obtenidos por Roig y col. (2014) (71 % para el estilo Pop) y los obtenidos por Schulze y Van Der Merwe (2010) (44 % para el estilo Burger).

2. Los resultados obtenidos para el estilo de Bach muestran un 66 % de expertos equivocados, similares a la media de los resultados obtenidos por Roig y col. (2014) (67 % de media) y superiores a los mejores resultados obtenidos por Schulze y Van Der Merwe (2010) (44 % para el estilo Burger).

3. Los resultados obtenidos para Palestrina y Monteverdi (45 % y 57 % de expertos equivocados, respectivamente), están por debajo de los peores resul-

tados de Roig y col. (2014) (67 % de media) y son similares o ligeramente superiores a los obtenidos por Schulze y Van Der Merwe (2010).

4. El estilo folk, entrenado con el corpus de Essen muestra unos malos resultados en cuanto al test de Turing, con un modesto 24 % de expertos equivocados, muy inferiores a la de los otros estudios.

Test de preferencia

Los resultados en cuanto al test de preferencia, encontramos resultados más modestos que en el test de Turing, siendo en general similares o peores que los encontrados en otros artículos. Profundizando encontramos que:

1. Los mejores resultados se encuentran en los estilos de Palestrina y Jazz (ambos con 53 % de preferencia de la composición humana sobre la computacional), en los que casi existe un empate entre el número de composiciones preferidas generadas por humano y generadas por ordenador.

2. En el estilo de Bach no se consigue llegar a un resultado tan bueno, ya que tan sólo el 28 % de las composiciones generadas por ordenador son favoritas frente a la generada por el humano.

3. Nuevamente, el estilo folk del corpus de Essen obtiene los peores resultados en cuanto a preferencia, con tan sólo una puntuación de un 21 %.

La explicación se puede encontrar en que en Schulze y Van Der Merwe (2010) se han utilizado composiciones procedentes de compositores humanos con poca experiencia compositiva, sin embargo aquí utilizamos como obras humanas las generadas por grandes figuras de la música. Este hecho puede distorsionar la comparativa. En particular, encontramos que el estilo de Bach es sumamente rico, complejo y cerrado, y el grupo de expertos lo conoce muy bien, ya que se estudia durante varios años en las asignaturas de contrapunto. Resulta muy complicado que el oído experto no encuentre un matiz melódico o armónico discordante con lo que se espera del estilo.

8.1.2 Valoraciones finales

El modelo de lenguaje para el estilo de Jazz ha funcionado sorprendentemente bien, posiblemente por sus características rítmicas y armónicas tan específicas de dicho lenguaje.

Bach es un estilo muy difícil de imitar, por sus complejas reglas armónicas; quizá mejoraría con órdenes de Markov más altos, pero posiblemente la composición se convertiría en una mera repetición sus fórmulas armónicas.

La explicación a los malos resultados de la emulación del estilo Folk se puede encontrar en las extrañas secuencias rítmicas que genera el sistema, que hacen al oyente experto detectar muy rápidamente el origen computacional de dichas producciones.

Nuestra población de individuos evaluadores se ha limitado a 43 individuos, sin embargo éstos no han sido escogidos de forma aleatoria: todos poseen una gran formación musical (músicos de Conservatorio Superior) y podrían ser considerados evaluadores con conocimiento experto, a diferencia los estudios de Roig y col. (2014) en el que se han utilizado 88 personas, elegidas aleatoriamente mediante encuesta pública, de las cuales un 17 % no tenía formación musical (no detallan el tipo de formación musical del 83 % restante), o el estudio de Schulze y Van Der Merwe (2010) en el que se han utilizado 263 individuos, seleccionados aleatoriamente entre personal y estudiantes y amigos de la Universidad de Stellenbosch (sin especificar tampoco el nivel de formación musical de los encuestados). Por tanto los resultados de nuestro sistema de evaluación no son completamente extrapolables a los realizados por estas dos investigaciones. En todo caso, nuestra evaluación ha sido más exigente a causa del alto nivel de formación musical de los encuestados. Sería conveniente repetir la encuesta con un grupo evaluador de control seleccionado aleatoriamente.

8.2 Limitaciones de la investigación

Hemos decidido acotar la presente investigación a las siguientes situaciones:

- Música tonal, relativa a los modos mayor y menor, excluyendo música atonal o música compuesta en otros modos como el dórico, frigio, etc.
- Por simplicidad, hemos excluido del análisis armónico la presencia de notas de paso o notas de adorno.

- No hemos abarcado el problema de la conducción melódica de las voces, ya que dicho problema constituye un problema de investigación en sí mismo.
- Nos hemos limitado únicamente a la utilización de bigramas y trigramas, sin abordar el uso de n-gramas de órdenes superiores $(4, 5, 6, \cdots)$ o muy superiores $(10, 11, \cdots)$.
- Hemos utilizado únicamente la técnica de smoothing de Laplace.
- La evaluación se ha realizado sin grupo de control seleccionado aleatoriamente, por lo que puede existir un sesgo en los resultados, ya que se han utilizado individuos evaluadores con una alta formación musical académica.

8.3 Futuras líneas de investigación

A continuación expondremos futuras líneas de investigación que aparecen una vez finalizado el presente trabajo:

- Extracción y utilización en los modelos markovianos de más características musicales, como la interválica existente en la melodía o la direccionalidad de las mismas.
- Utilización de órdenes de Markov superiores y distintos métodos de suavizado.
- Diseño de un experimento para la evaluación del material musical generado por nuestro sistema en el que exista un grupo de control de evaluadores, seleccionados aleatoriamente, confiriendo al diseño evaluador una mayor validez experimental.
- Diseño de una ontología armónica que incorpore las restricciones de la armonía clásica, para la armonización de los acordes generados.
- Ampliación del sistema a otro tipo de estéticas musicales: música atonal, o música modal, música Pop.
- Implementación de nuevas restricciones rítmicas para mejorar los ritmos generados en algunos estilos, como el caso del estilo Folk de Essen.
- Diseñar una ontología, o ampliar la existente, modelando los distintos estados emocionales con relaciones entre distintas características musicales. De esta manera, una vez implementada en nuestro sistema, el usuario compositor podría configurar la generación de música relativa a un determinado

estado anímico[1], abriendo las puertas a la integración de nuestro proyecto con sistemas empáticos, capaces de detectar el estado anímico del usuario y generar música en función del mismo.

[1] Véase el artículo Empathy and embodied experience in virtual environment: To what extent can virtual reality stimulate empathy and embodied experience? de Shin (2018) o el artículo Learning empathy through virtual reality: multiple strategies for training empathy-related abilities using body ownership illusions in embodied virtual reality de Bertrand y col. (2018).

Bibliografía

A. Prechtl R. Laney, A. W. y Samuels, R. (2014). Algorithmic music as intelligent game music. *Proceedings of the AISB Anniversary Convention*. New York, USA, págs. 1-4 (vid. pág. 20).

Abel, J. F. (1981). *Computer Composition of Melodic Deep Structures*. Ann Arbor, Michigan: Michigan Publishing, University of Michigan Library (vid. págs. 18, 23).

Allan, M (2002). Harmonising chorales in the style of Johann Sebastian Bach. Tesis doct. School of Informatics, University of Edinburgh (vid. pág. 20).

Alvira, J. R. (2009). *Cifrados barrocos de acordes de sÃ©ptima*. URL: `https://www.teoria.com` (visitado 22-04-2019) (vid. pág. 68).

Ariza, C. (2011). Two pioneering projects from the early history of computer-aided algorithmic composition. *Computer Music Journal*, 35(3), págs. 40-56 (vid. pág. 13).

Bas, J. y Lamuraglia, N. (1977). *Tratado de la forma musical*. Madrid, España: Ricordi Americana (vid. pág. 55).

Basharin, G. P., Langville, A. N. y Naumov, V. A. (2004). The life and work of AA Markov. *Linear algebra and its applications*, 386, págs. 3-26 (vid. pág. 41).

Bertrand, P., Guegan, J., Robieux, L., McCall, C. A. y Zenasni, F. (2018). Learning empathy through virtual reality: multiple strategies for training empathy-related abilities using body ownership illusions in embodied virtual reality. *Frontiers in Robotics and AI*, 5, pág. 26 (vid. pág. 99).

Bird, S. y Loper, E. (2019). *Natural Language Toolkit*. URL: `https://www.nltk.org/` (visitado 11-03-2019) (vid. pág. 48).

Bronstein, I. N., Hromkovic, J., Luderer, B., Schwarz, H.-R., Blath, J., Schied, A., Dempe, S., Wanka, G. y Gottwald, S. (2012). *Taschenbuch der mathematik*. Vol. 1. Oxford, UK: Springer-Verlag (vid. pág. 42).

Brooks Jr, F. P., Hopkins Jr, A. L., Neumann, P. G. y Wright, W. V. (1957). *An experiment in musical composition*. Cambridge, Massachusetts: The MIT Press, págs. 23-40 (vid. págs. 13, 14).

Browne, T. M. y Fox, C. (2009). Global expectation-violation as fitness function in evolutionary composition. *Proceedings of Workshops on Applications of Evolutionary Computation*. Tubingen, Germany, págs. 538-546 (vid. pág. 16).

Buys, J. y Merwe, B. van der (2012). Chorale harmonization with weighted finite-state transducers. *Proceedings of the Twenty-Third Annual Symposium of the Pattern Recognition Association of South Africa*. Ciudad del Cabo, South Africa, págs. 95-101 (vid. pág. 3).

Cherfi, S., Hamdi, F., Rigaux, P., Thion, V. y Travers, N. (2017). Formalizing Quality Rules on Music Notation–an Ontology-based Approach. *Proceedings of the International Conference on Technologies for Music Notation and Representation (TENOR17)*. A Coruña, España (vid. pág. 6).

Cherla, S., Purwins, H. y Marchini, M. (2013). Automatic phrase continuation from guitar and bass melodies. *Computer Music Journal*, 37(3), págs. 68-81 (vid. págs. 21, 23).

Chuan, C.-H. y Chew, E. (2011). Generating and evaluating musical harmonizations that emulate style. *Computer Music Journal*, 35(4), págs. 64-82 (vid. pág. 20).

Coca, A. E., Romero, R. A. y Zhao, L. (2011). Generation of composed musical structures through recurrent neural networks based on chaotic inspiration.

Proceedings of The 2011 International Joint Conference on Neural Networks. San Jose, California, USA, págs. 3220-3226 (vid. pág. 16).

Contreras, R. E. *Inversión de un acorde* (vid. pág. 68).

Cruz-Alcázar, P. P. y Vidal-Ruiz, E. (1998). Learning regular grammars to model musical style: Comparing different coding schemes. *International Colloquium on Grammatical Inference*. Valencia, España, págs. 211-222 (vid. págs. 19, 23).

Cuthbert, M., Ariza, C., Hogue, B. y Oberholtzer, J. W. (2019). *music21: a toolkit for computed-aided musicology*. URL: `https://web.mit.edu/music21/` (visitado 16-09-2018) (vid. pág. 9).

Cuthbert, M. S. y Ariza, C. (2010). music21: A toolkit for computer-aided musicology and symbolic music data. *Proceedings of the 11th International Society for Music Information Retrieval Conference*. Utrecht, Netherlands: International Society for Music Information Retrieval (vid. pág. 48).

Cuthbert, M. S., Ariza, C. y Friedland, L. (2011). Feature Extraction and Machine Learning on Symbolic Music using the music21 Toolkit. *Proceedings of the 12th International Society for Music Information Retrieval Conference*. Miami, Florida, págs. 387-392 (vid. pág. 21).

Cuthbert, M. S., Ariza, C., Cabal, J., Hadley, B. y Parikh, N. (2011). Hidden Beyond MIDI Reach: Feature Extraction and Machine Learning with Rich Symbolic Formats in music21. *Proceedings of the Neural Information Processing Systems Conference*. Granada, España (vid. pág. 49).

Dahlstedt, P. (2007). Autonomous evolution of complete piano pieces and performances. *Proceedings of Music AI Workshop*. Hyderabad, India (vid. pág. 17).

Downie, J. S. (2019). *The Music Information Retrieval Evaluation eXchange*. URL: `https://www.music-ir.org/mirex/wiki/MIREX_HOME` (visitado 20-04-2019) (vid. pág. 9).

Falkenstein, J. T. von y Tlalim, T. (2009). Alter Ego: A Generative Music Creation System. *Proceedings of the 2009 International Computer Music Conference*. Ann Arbor, Michigan (vid. pág. 17).

Farbood, M. y Schöner, B. (2001). Analysis and Synthesis of Palestrina-Style Counterpoint Using Markov Chains. *Proceedings of 2001 International Computer Music Conference*. Havana, Cuba (vid. pág. 19).

Fazekas, G. y Tidhar, D. (2009). *Notes on Music Information Retrieval*. URL: `http://dbtune.org/onto/tm/doc/temperament.html` (visitado 13-06-2019) (vid. pág. 57).

Fernández-López, M., Gómez-Pérez, A. y Juristo, N. (1997). Methontology: from ontological art towards ontological engineering. *Proceedings of the Ontological Engineering AAAI-97*. Stanford University, USA: American Asociation for Artificial Intelligence (vid. págs. 7, 54).

Gelbukh, A., Sidorov, G. y Velásquez, F. (2003). Análisis morfológico automático del español a través de generación. *Revista Escritos*, 28, págs. 9-26 (vid. pág. 34).

Good, M. (2001a). MusicXML for notation and analysis. *The virtual score: representation, retrieval, restoration*, 12, págs. 113-124 (vid. págs. 8, 49).

— (2001b). MusicXML for notation and analysis. *The virtual score: representation, retrieval, restoration*, 12, págs. 113-124 (vid. pág. 49).

Harley, N. y Wiggins, G. (2015). An ontology for abstract, hierarchical music representation. *Proceedings of the 16th International Society for Music Information Retrieval Conference ISMIR 2015*. Málaga, España, pág. 12 (vid. pág. 6).

Hild, H., Feulner, J. y Menzel, W. (1992). HARMONET: A neural net for harmonizing chorales in the style of J.S. Bach. En: *Advances in Neural Information Processing Systems*. Ed. por J. E. Moody, S. J. Hanson y R. P. Lippmann. Burlington, Massachusetts: Morgan-Kaufmann, págs. 267-274 (vid. pág. 16).

Hiller, L. (1959). Computer music. *Scientific American*, 201(6), págs. 109-121 (vid. pág. 15).

— (1968). *Music composed with computers: an historical survey*. 18. Urbana-Champaign, USA: University of Illinois (vid. pág. 13).

Hiller, L. e Isaacson, L. (1959). *Experimental music: composition with an electronic computer*. New York, USA: McGraw-Hill (vid. págs. 13, 15).

Hirzel, M. y Soukup, D. (2000). *Project Writeup for CSCI 5832 natural language processing*. Colorado, USA: University of Colorado (vid. pág. 19).

Holmes, T. (2012). *Electronic and experimental music: technology, music, and culture*. New York, USA: Routledge (vid. pág. 14).

Jensen, J. H. (2011). Evolutionary music composition: A quantitative approach. Tesis de mtría. Trondheim, Norwey: Institutt for datateknikk og informasjonsvitenskap (vid. pág. 17).

Jones, J., Siqueira Braga, D. de, Tertuliano, K. y Kauppinen, T. (2017). MusicOWL: the music score ontology. *Proceedings of the International Conference on Web Intelligence*. Leipzig, Germany, págs. 1222-1229 (vid. pág. 58).

Jones, K. J. (1980). Sound Generating Techniques on the ITT 2020 and Apple II Computers. *Proceedings of the Conference on Computer Music in Britain*. London, UK (vid. págs. 18, 23).

Keller, R. M. y Morrison, D. R. (2007). A grammatical approach to automatic improvisation. *Proceedings of Fourth Sound and Music Conference*. Lefkada, Greece (vid. pág. 17).

Kleene, S. C. (1956). *Representation of events in nerve sets, in .ªutomata studies"*. Princeton, USA: Princeton Univ. Press (vid. pág. 28).

Klein, M. (1957). Syncopation in Automation. *Radio-Electronics*, 1, págs. 36-37 (vid. pág. 13).

Knab, B. (2000). Erweiterung von Hidden-Markov-Modellen zur Analyse ökonomischer Zeitreihen. Tesis doct. Universität zu Köln (vid. pág. 44).

Liu, C. H. y Ting, C. K. (2017). Computational intelligence in music composition: A survey. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 1(1), págs. 2-15 (vid. pág. 20).

Madnani, N. (2007). Getting started on natural language processing with Python. *ACM Crossroads*, 13(4), pág. 5 (vid. pág. 48).

Manaris, B. y Hughes, D. (2011). Monterey mirror: combining Markov models, genetic algorithms, and power laws. *Proceedings of 1st Workshop in Evolutionary Music, 2011 IEEE Congress on Evolutionary Computation (CEC 2011)*. Auburn, USA, págs. 33-40 (vid. págs. 21, 23).

Martin, J. H. y Jurafsky, D. (2009). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. London, UK: Pearson/Prentice Hall Upper Saddle River (vid. págs. 28-36, 38-40, 66).

Mauco, V. y Barbuzza, R. (2009). *Apuntes de Ciencias de la Computación*. Inf. téc. Facultad de ciencias exactas. Universidad Nacional del Centro de la Provincia de Buenos Aires. (vid. págs. 30, 31).

McVicar, M., Fukayama, S. y Goto, M. (2014). AutoRhythmGuitar: Computer-aided composition for rhythm guitar in the tab space. *Proceedings of the 2014 International Computer Music Conference*. Athens, Greece (vid. págs. 22, 23).

Meinecke, J. (1981). *Stochastic Melody Writing Procedures: An Analysis Based Approach*. Ann Arbor, Michigan: Michigan Publishing, University of Michigan Library (vid. pág. 18).

Mohri, M. (1996). On some applications of finite-state automata theory to natural language processing. *Natural Language Engineering*, 2(1), págs. 61-80 (vid. pág. 34).

Mozer, M. C. (1994). Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing. *Connection Science*, 6(2), págs. 247-280 (vid. pág. 16).

Musescore.org (2019). URL: `https://musescore.org/es` (visitado 15-04-2019) (vid. pág. 50).

musicXML (2019). URL: `https://en.wikipedia.org/wiki/MusicXML` (visitado 09-04-2019) (vid. pág. 50).

Nierhaus, G. (2009). *Algorithmic composition: paradigms of automated music generation*. New York, USA: Springer Science & Business Media (vid. págs. 2, 18-20, 41-44).

Pachet, F., Roy, P. y Barbieri, G. (2011). Finite-length Markov processes with constraints. *Proceedings of Twenty-Second International Joint Conference on Artificial Intelligence*. Barcelona, España (vid. págs. 21, 23).

Piston, W. (1989). *Armonía*. Barcelona: Editorial Labor (vid. pág. 7).

PyScripter (2019). URL: `https://en.wikipedia.org/wiki/PyScripter` (visitado 09-04-2019) (vid. pág. 46).

Quick, D. (2010). *Generating music using concepts from Schenkerian analysis and chord spaces*. Inf. téc. Yale University (vid. pág. 17).

Raimond, Y., Abdallah, S. A., Sandler, M. B. y Giasson, F. (2007). The Music Ontology. *Proceedings of 8th International Conference on Music Information Retrieval*. Vol. 2007. Philadelphia, USA (vid. pág. 56).

Rashid, S. M., De Roure, D. y McGuinness, D. L. (2018). A music theory ontology. *Proceedings of the 1st International Workshop on Semantic Applications for Audio and Music*. Monterey, USA, págs. 6-14 (vid. pág. 6).

Roig, C., Tardón, L. J., Barbancho, I. y Barbancho, A. M. (2014). Automatic melody composition based on a probabilistic model of music style and harmonic rules. *Knowledge-Based Systems*, 71, págs. 419-434 (vid. págs. 3, 22, 23, 87, 91, 92, 95-97).

Schaffrath, H. (1995). The Essen folksong collection. *Database containing*, 6, págs. 15-17 (vid. pág. 66).

Schonberg, A. (1987). *Fundamentos de la composición musical*. Madrid, España: Real Musical (vid. pág. 7).

Schulze, W. y Van Der Merwe, B. (2010). Music generation with Markov models. *IEEE MultiMedia*, 3, págs. 78-85 (vid. págs. 2, 20, 23, 87, 91, 92, 95-97).

Scott, M. (2019). *music21 User Guide*. URL: `http://web.mit.edu/music21/doc/usersGuide/` (visitado 22-04-2019) (vid. pág. 69).

Sertan, S. y Chordia, P. (1975). Modeling melodic improvisation in Turkish folk music using variable-length Markov models. *Proceedings of the 12th Interna-*

tional Society for Music Information Retrieval Conference. Urbana, Illionis, págs. 269-274 (vid. pág. 20).

Shin, D. (2018). Empathy and embodied experience in virtual environment: To what extent can virtual reality stimulate empathy and embodied experience? *Computers in Human Behavior*, 78, págs. 64-73 (vid. pág. 99).

Stevens, R. (2016). *The Protégé wiki*. URL: `https://protegewiki.stanford.edu/` (visitado 13-06-2019) (vid. pág. 47).

Supper, M. (2004). *Música electrónica y música con ordenador: historia, estética, métodos, sistemas*. Madrid, España: Alianza Editorial (vid. pág. 2).

Tipei, S. (1975). MP1: a computer program for music composition. *Proceedings of the Second Music Computation Conference*. Urbana, Illionis, págs. 68-82 (vid. págs. 15, 23).

Tjoa, S. (2019). *Notes on Music Information Retrieval*. URL: `https://musicinformationretriev com/about.html` (visitado 20-04-2019) (vid. pág. 9).

Toch, E. (1989). *La melodía*. Barcelona, España: Idea Books (vid. pág. 7).

Triviño-Rodriguez, J. L. y Morales-Bueno, R. (2001). Using multiattribute prediction suffix graphs to predict and generate music. *Computer Music Journal*, 25(3), págs. 62-79 (vid. pág. 19).

Wagner, R. A. y Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1), págs. 168-173 (vid. pág. 35).

Web Ontology Language (2019). URL: `https://en.wikipedia.org/wiki/Web_Ontology_Language` (visitado 13-06-2019) (vid. pág. 47).

Zamacois, J. (1979). *Tratado de Armonía*. Vol. 1. Barcelona, España: Editorial Labor (vid. pág. 7).

— (2002). *Tradato de Armonía*. Barcelona, España: Idea books (vid. pág. 55).

Zhang, Y. (2015). *An Introduction to Python and computer programming*. New York, USA: Springer, págs. 1-11 (vid. pág. 46).

Zheng, X., Li, D., Wang, L., Shen, L., Gao, Y. y Zhu, Y. (2017). An automatic composition model of Chinese folk music. *Proceedings of AIP Conference*. Vol. 1820. 1. Bydgoszcz, Poland (vid. págs. 3, 22, 23).

Zhu, H., Liu, Q., Yuan, N. J., Qin, C., Li, J., Zhang, K., Zhou, G., Wei, F., Xu, Y. y Chen, E. (2018). Xiaoice band: A melody and arrangement generation framework for pop music. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. London, UK, págs. 2837-2846 (vid. págs. 22, 23).