Universitat d'Alacant
Universidad de Alicante

Escola Politècnica Superior
Escuela Politécnica Superior

Proyecto Fin de Carrera
Ingeniería Técnica de Telecomunicación
Sonido e Imagen

Detección de armonía y estructura rítmica musical

Autor:
Gabriel Meseguer Brocal

Tutor:
Antonio Pertusa Ibáñez

Junio, Año 2009

Í N D I C E

1 . Introducción .

“ Information is not knowledge,
Knowledge is not wisdom,
Wisdom is not truth,
Truth is not beauty,
Beauty is not love,
Love is not music and
Music is THE BEST ”

Frank Zappa, 1979

1.0. Prefacio.

La música, como todo arte, está en constante desarrollo. Durante todo su historia siempre ha habido genios incomprendidos que han tirado del triángulo kandiskyano hacia arriba haciendo accesible, con el paso del tiempo, sus innovadores conocimientos a los escalafones mas bajos. Hoy día ya no existe ese triángulo como tal, las innumerables corrientes musicales que se han venido desarrollando desde el siglo XX hasta nuestros días han hecho que cada una tenga su propio triángulo. No solo en la música formal; donde los diferentes movimientos aparecidos durante el siglo XX y XXI han experimentado con todas las características musicales; si no que también se ha incrementado de manera notable la variedad y calidad de la música popular gracias en una parte muy importante a los avances en la comunicación permitiendo la mezclas de géneros hasta ahora aislados. A pesar de todo hoy en día no parece existir ninguna figura en la cúspide del triángulo (aunque bien es sabido que estos genios suelen ser desconocidos por su coetáneos) que haga avanzarlo hacia una dirección concreta. El problema está en que la proceso actual no se fundamenta tanto en nombres como en instrumentos. En este campo la mayor revolución sin lugar a dudas es el ordenador. Su aparición rompe todos los esquemas y amplia inimaginablemente el campo de investigación y actuación. Permite en un único aparato crear nuevos instrumentos y facilitar funciones tales como la escritura musical, su análisis e investigación. Todo esto ha supuesto un cambio en la problemática que ya no solo consta de parámetros como ritmo, textura, timbre, forma o altura y demás parámetros tradicionales, ya que parámetros más técnicos como son formas de onda, espectros, parciales armónicos, ecualización o síntesis de audio cobran gran importancia.

Dentro de este nuevo campo la parte encargada de la recuperación de información musical (Music Information Retrieval - MIR) es muy importante puesto que engloba: 1) métodos de clasificación, agrupación y moldeado de canciones en función de los parámetros extraídos del audio musical; 2) métodos para la identificación musical, bases de datos y software de recuperación musical; 3) interfaces que permiten la interacción entre el hombre y el ordenador; 4) percepción musical, cognición, afección y emociones; 5) análisis musical y representación de los conocimientos para su posterior uso en software.

Todo esto da a MIR una gran importancia puesto que ayuda en los estudios de mercado de la industria musical, en la identificación de la propiedad intelectual (en caso de plagios) y sobre todo da facilidades para el músico a la hora de desarrollar sus estudios e investigaciones.

Un punto muy importante dentro del MIR es la correcta extracción de características musicales (armonía, tonalidad, métrica, tempo, compás) puesto que es el inicio de todo el proceso y la base de los demás puntos. Estas características se pueden clasificar en dos grandes grupos: rítmicas y armónicas. De todas, el tempo, entendido como la velocidad de ejecución de una pieza, es uno de los datos más importantes, ya que representa "la unidad de medida" básica en la música. Existen multitud de sistemas de detección automática de tempo, la mayoría basados en cambios energéticos de la señal. Estos sistemas consiguen buenos resultados con melodías que tienen elementos percusivos pero cometen bastantes errores con señales que no presentan fuertes cambios en su energía. Por otro lado, la información armónica se puede extraer sin tener en cuenta la información rítmica (tempo, métrica, compás...), aunque la mayoría de los sistemas que tratan de detectar la armonía infieren primero el tempo basándose en cambios de energía, tras lo cual se realiza el análisis armónico. El conocimiento de la armonía permite calcular la tonalidad de una canción así como su clasificación dentro de un género u otro.

1.1. Objetivo del proyecto.

Lo que se plantea en este proyecto es construir un sistema automático de extracción de información armónica y rítmica de una señal de audio musical. Por norma general, estos dos procesos se suelen realizar de manera independiente. Lo mas común es detectar la armonía usando información rítmica calculada previamente y calcular la información rítmica usando canciones con la armonía ya etiquetada correctamente o usando sistemas de detección de tempo basados en cambios de energía.

Este proyecto une estos dos campos en un punto intermedio, donde ambos se complementan. Para ello se invierte el orden: 1º extracción de información armónica, 2º extracción de información rítmica (frente a 1º información rítmica 2º información armónica, tradicional) partiendo de la premisa: "la duración de los paquetes armónicos es proporcional al tempo de dicha canción". Por tanto primero se detectarán los cambios de armonía lo que permitirá calcular la duración de cada uno y posteriormente se calculará el tempo. Además, como se obtendrá un análisis armónico de la canción, se calculará la tonalidad y se clasificará la canción en el género correspondiente.

Para ello se realizará un software que sea capaz de detectar la armonía de una canción sin conocer la información rítmica, para calcular el tempo, así como la tonalidad y clasificar su género. El proyecto se dividirá en dos fases:

1º Desarrollo con música simbólica (partituras electrónicas en formato MIDI) y experimentación de los resultados obtenidos comparándolos con otros trabajos similares.

2º Desarrollo con señales de audio usando como enlace el sistema de transcripción polifónica de Antonio Pertusa y José M. Iñesta (2008). Este sistema se encargará de traducir la señal de audio en un fichero MIDI de manera que ambos sistemas se combinen para poder extraer la información armónica y rítmica de audio de la mejor manera posible.

1.2. Conceptos básicos.

1.2.1 Conceptos musicales.

Sonidos, sucesivos o simultáneos, distintos, parecidos, iguales, aislados o en multitud, … todos van a constituir la música, pero el modelo clásico ha establecido como elementos esenciales:

■ El ritmo. Es entendido a pequeña escala como la combinación de duraciones y acentos de sonidos (determinados o indeterminados). Con frecuencia se establecen motivos rítmicos que se repiten configurando una obra. Pero la variedad de planos en los que se producen acentuaciones y su contribución a la tensión musical[1] nos hace pensar en el ritmo de una manera más general, entendido como las combinaciones cambiantes de tiempo e intensidad de todos los elementos (melodía, armonía, sonoridad, el tempo y las interacciones de los distintos elementos) de una obra musical.

Especialmente interesante es el ritmo armónico o periodicidad de los cambios de armonía, que pueden constituirse en un modelo con las dimensiones de una frase o una sección.

De entre todos los parámetros rítmicos destaca el pulso, entendido en el renacimiento como tactus, beat en la música actual y que equivale al latir de la música, unidad que ordena acentos y silencios, y subyace en las obras musicales[2]. La pulsación puede ser acentuada o no acentuada. La combinación de pulsos acentuados y no acentuados da lugar a los compases, que fundamentalmente se pueden agrupar en binarios (fuerte-débil), ternarios (fuerte-débil-débil) y cuaternario (fuerte-débil-semifuerte-débil). Desde la polifonía, la necesidad de simultánear sonidos llevó a la búsqueda de procedimientos de notación rítmica hasta alcanzar al sistema clásico en donde el valor de cada sonido es representado mediante un símbolo:

---

[1] La tensión musical es el momento en el que los elementos musicales se encuentran en máxima actividad. Alterna con momentos de relajación de todos los elementos o de uno en tanto que los demás continúan su curso.

[2] Movimiento intuitivo de pie o mano que se realiza al escuchar una pieza musical.

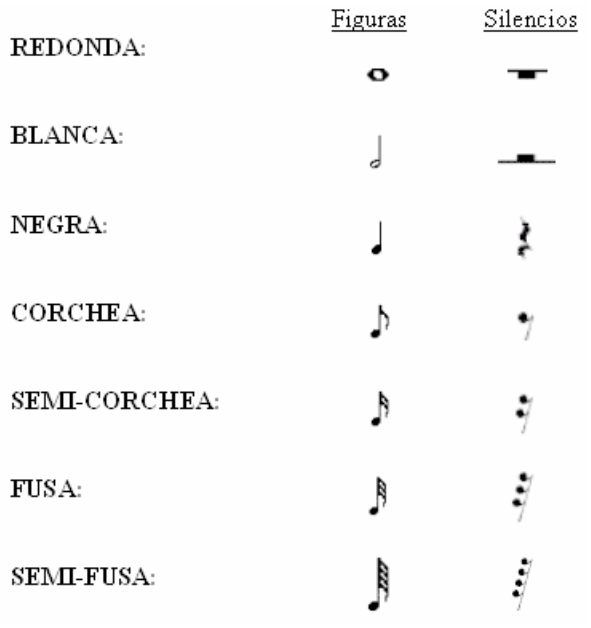


Figura 1: Duraciones musicales.

Las relaciones que mantiene entre sí están matizadas por los compases que se expresan mediante fracciones en donde el denominador hace referencia a la unidad de medida (1 representa una redonda, 2 una blanca, 4 una negra y así sucesivamente siguiendo el esquema anterior) y el numerador indica el número de unidades que componen cada compás y la distribución de acentos:

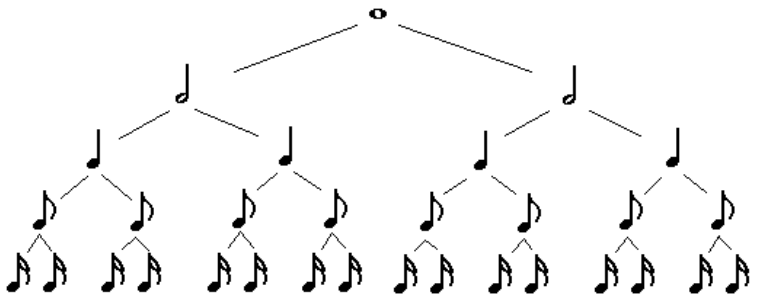


Figura 2: relación entre duraciones musicales.

■ La melodía. Sucesión de sonidos de distinta altura con libertad de movimiento que busca alcanzar tensión y relajación, animada por el ritmo, se extiende y contrae dotada de flexibilidad.

La melodía ha sido el elemento básico de la música de tradición occidental, ya sea sola, simultáneamente con otras líneas melódicas (contrapunto) o en recíproco influjo con la armonía (simultaneidad sonora) pues muchas melodías está íntimamente relacionadas con ésta, y las relaciones entre ellas pueden oscilar entre:

- Un cambio rápido de la melodía provoca gran variedad de acordes diferentes o al contrario, una simplificación armónica base de los cambios melódicos.
- La rápida evolución de la armonía puede hacer que la melodía se mueva en progresiones o que dé lugar a novedades melódicas inesperadas.

Breve evolución histórica de la armonía.

A partir del establecimiento del principio de la melodía acompañada barroca, el uso de acordes[3] se generalizó, pero aún faltaba un principio para que se estableciera la armonía como un sistema jerarquizado :

■ La afinación temperada, que divide la octava en doce partes iguales, o semitonos, posibilitaba el uso de todos los acordes y la modulación[4] sin límites:

■ do, do#/re b, re, re#/mi b, mi, fa, fa#/sol b, sol, sol#/la b, la, la#/si b, si[5].

A lo largo de la historia y las culturas se han establecido diferentes escalas u ordenaciones de sonidos (pentatónica, heptáfona). En el Renacimiento y el Barroco, se implantó como base la escala heptáfona[6] que se divide en siete grados:

[3] Dos o más sonidos simultáneos, el acorde triada, tres sonidos por terceras, es el más usado.

[4] Modulación: cambio de una jerarquía de acordes a otra.

[5] A partir de ahora se usará la notación internacional: C, C#/Db, D, D#/Eb, E, F, F#/Gb, G, G#/Ab, A, A#/Bb, B.

[6] Se utiliza el modelo con notas naturales.

Do	Re	Mi	Fa	Sol	La	Si	Do
I	II	III	IV	V	VI	VII	I

Figura 3: escala natural.

recibiendo los mas importantes, en función de su poder de atracción o alejamiento sobre el resto, los siguientes nombres: Iº o tónica, IV o subdominante y V o dominante.

Desde el Renacimiento, el movimiento conclusivo o cadencia de quinta en el bajo[7] fue entendido como el más "natural", y posteriormente constituyó la cadencia perfecta Vº Iº. Los compositores barrocos vieron que para definir la tonalidad[8] no bastaban únicamente dos acordes, sino que se necesitan tres, como mínimo:

- Tónica (T) o acorde sobre el grado I , que contiene las notas 1ª, 3ª y 5ª, y constituye el centro tonal.
- Dominante (D) o acorde sobre el grado V, que contiene las notas 5ª, 7ª y 2ª, que imprime tensión.
- Subdominante (Sd) o acorde sobre el grado IV, con las notas 4ª, 6ª y 8ª = 1ª o también el acorde sobre el grado IIº, con las notas 2ª, 4ª y 6ª, que impulsan un alejamiento distendido.

Charpentier, Vivaldi, Händel,…ejemplifican a la perfección la serie de acordes
I, IV, I, V, I (T, Sd, T, D, T), lo que es mas bien una excepción en Bach, fragmentos de corales, pues en él la situación armónica es más complicada.

El Barroco estableció nuevos acordes:

- Acorde de séptima de dominante en las cadencias (5º, 7º, 2º y 4º).
- Acorde con sexta añadida (4º, 6º, 8º y 9º = 2º).

El clasicismo supuso la clarificación y el triunfo definitivo de los procesos armónicos:

[7] Linea melódica más grabe que condiciona en gran medida la armonía.

[8] Jerarquización de sonidos en torno a una nota/acorde principal o tónica.

- El bajo, inicialmente, se limitará a las tres funciones principales (T, Sd y D)
- La melódica clásica se liga íntimamente al acompañamiento y sobre todo a la cadencia.
- Un fraseo claro y equilibrado con constantes repeticiones crea el ritmo armónico.
- Un tempo más vivo da lugar a menor número de cambios armónicos.

Algunos acordes que aportó el clasicismo:

- Séptima sobre el séptimo grado en el modo mayor (7º, 2º, 4º y 6º) que mezcla las funciones de dominante y subdominante.

El Romanticismo hará de la armonía un ámbito de inspiración, multiplicándose las posibilidades en situaciones concretas de composición, pero a la vez su culto a lo natural, el genio como ausencia de sofisticación y la búsqueda de la esencia musical de los pueblos, revertirán en la sencillez de la melodía, el equilibrio del fraseo y la armonía, sobre todo presentes en las pequeñas formas ternarias, canciones sin palabras, barcarolas, impromptus, momentos musicales, lied, … que intenta reflejar la sencillez popular.

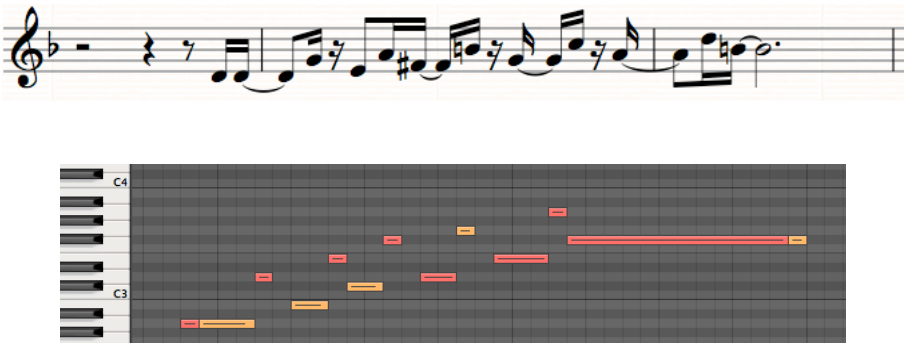
En el siglo XX la música culta se adentra en la experimentación y se aleja del público, aparecen modelos abstractos que crean sus propios sistemas de organización, dodecafonía, serialismo, indeterminación y aleatoriedad, música textural… La mayoría se definen por su contraposición a los preceptos y parámetros tradicionales, por lo que la tonalidad y la armonía funcional caen en desuso, pero a la vez mucha música de consumo, para bailar, comercial, popular, folk, jazz, blues, pop, o rock, mantiene los esquemas tradicionales, con determinadas peculiaridades, en donde de nuevo la tonalidad y la armonía desempeñan una decisiva función en la organización de la obra musical.

1.2.2. Otro conceptos.

Pianola.

Una pianola es un instrumento musical, que tiene como base el piano, y al que se le añaden un conjunto de elementos mecánicos y neumáticos que permiten la reproducción automática de la música. Se controla mediante unas rollos de papel con unos agujeros en donde la altura relativa de cada agujero corresponde con la altura de cada nota, y la longitud del mismo con la duración.

La representación pianola simula los rollos de papel usados en este instrumento. En ella las notas se representadas gráficamente como rectángulos de longitud variable. El tiempo transcurre horizontalmente y de izquierda a derecha, mientras que la posición vertical de cada cuadro determinada la altura de cada nota (normalmente viene acompañada de un piano que define el valor de cada altura). Esta representación es muy sencilla de asimilar y permite entender la notación musical tradicional si necesidad de conocerla profundamente. En la siguiente figura se muestra un ejemplo de partitura y su correspondiente pianola:



MIDI.

El Midi es un protocolo estándar de intercambio de información entre ordenadores, sintetizadores, secuenciadores, controladores y otros dispositivos musicales electrónicos. La directiva MIDI se refiere al protocolo de comunicación entre los dispositivos, los conectores, la manera de transmitir los mensajes (Serie, asíncrona, 32.150 bits/s), el tipo de mensajes y su codificación. El formato MIDI se suele usar para representar partitura en el ordenador.

2. Estado de la cuestión.

2.1. Antecedentes.

El tempo es un elemento muy importante dentro de la música occidental puesto que se trata de la unidad básica de medida de una pieza y condiciona la duración de todas las notas. Determina la progresión horizontal de una canción. Tener conocimiento del tempo permite realizar tareas tales como sincronización de mezclas en base al tempo, división de una canción en unidades, u obtener información necesaria para el análisis musical (interpretación, frases, estructuras, repeticiones...) Por todo esto, la estimación del tempo es un campo importante dentro del MIR (Music Information Retrieval), centrando esfuerzos en desarrollar sistemas capaces de detectar el tempo de la mejor manera posible. Aún así es un problema que no está resuelto, sobre todo a la hora de abordar la música clásica tonal y el jazz.

Los métodos de estimación del tempo pueden agruparse aproximadamente en dos líneas de investigación diferentes, todas ellas enmarcadas dentro del cálculo del tempo mediante onsets[9]:

1) Métodos que usan la energía de la señal a lo largo del tiempo (normalmente usando un banco de filtros) para medir la periodicidad de esta, como es el caso de Eric D. Scheirer (1998).

2) Métodos que determinan de entre todos los onsets de una canción cuales puede ser de beat/pulso . Con estos onsets se construye un histograma que almacena las diferencias interválicas entre ellos en segundos (IOI: Inter-Onset-Interval), buscando los de mayor periodicidad. Dentro de este marco se engloban los trabajos de Simon Dixon (2001) y Fabien Gouyon (2002). A la hora discriminar onsets normalmente se tiene en cuenta la energía de cada uno de ellos, como en el trabajo de Geoffroy Peeters (2005), que optimiza el sistema usando el flujo de energía espectral mediante una combinación de la Transformada de Fourier y la función de

[9] Onset se refiere al inicio de una nota

autocorrelación mapeada en frecuencia, para abordar tanto los casos en donde la percusión esta presente como en los que no. También existen aproximaciones que se basan en el análisis de la percepción psicoacústica ,como son los estudios de Frank Seifert, Katharina Rasch y Michael Rentzsch (2006) en donde definen una serie de normas en base al análisis de conceptos de percepción que disciernen qué onsets son pulsos. También existen otras aproximaciones como las de Ruohua Zou y Joshua D.Reiss (2007), que plantean clasificar los onsets en fuertes y débiles siguiendo métodos distintos para cada caso. Para detectar los onsets fuertes usan cambios de energía mientras que para los débiles desarrollan un algoritmo que se basa en la altura de las notas.

En todos los casos anteriormente mencionados, el tempo trata de calcularse en base a características puramente energéticas, puesto que la información armónica no tiene ninguna importancia (jerarquía que tienen las notas analizadas, análisis de sus alrededores...) a excepción de Ruohua Zou y Joshua D.Reiss (2007) en donde si se tiene en cuenta, pero sólo para calcular los onsets débiles.

Si el problema del tempo se afronta desde una óptica musical, la armonía tiene mucho que decir, puesto que si el tempo es la unidad básica de la música en notación horizontal, la armonía lo es en notación vertical y van muy ligadas. No se ha encontrado ningún trabajo previo que trate de calcular el tempo en base a la armonía.

Sin embargo, sí que existen multitud de artículos que afrontan la detección armónica. En general, el problema se afronta con el tempo ya detectado como en las investigaciones de David Temperley y Daniel Sleator (1999), o los de Heinrich Taube (1999); o bien se calcula en base a cambios de energía como en David Temperley y Daniel Sleator (2004). Por el contrario, en los trabajos de Bryan Pardo y William P. Birminghan (2002) se plantea la posibilidad de segmentar una canción en función de la armonía (sin necesidad de conocer ningún parámetro rítmico) y a su vez realizar el análisis armónico.

2.2. Planteamiento del problema.

En los artículos anteriormente mencionados, el tempo y la armonía son considerados problemáticas diferentes: El tempo se calcula en función de los onsets y para afrontar el análisis armónico es necesario conocer el valor del tempo. Sin embargo podemos calcular la evolución armónica de una canción y suponer, en base a los conceptos de armonía conocidos, que entre la armonía y el tempo existe algún tipo de relación lo que permitirá calcular el tempo sin necesidad de conocer la energía de los onsets o información del tipo tiempo de ataque, tiempo débil, fuerte semi-fuerte...

En este proyecto se variará el orden de actuación a la hora de afrontar el problema. Primero se calculará la armonía tras lo cual se podrá conocer la duración de cada paquete armónico. Si se relaciona el paquete armónico que más se repita con el pulso se podrá calcular el tempo. Esta suposición no es cierta del todo, puesto que por regla genera general, en canciones con tempo rápido la armonía varía cada compás y no cada tempo, mientras que en canciones con el tempo muy lento la armonía puede variar en instantes de tiempo inferiores a la unidad, normalmente la mitad o un cuarto. Aún así la evolución armónica en la música occidental siempre suele variar en múltiplos o submúltiplos enteros del tempo. Por lo que un conocimiento del la estructura armónica de una canción puede permitir calcular el tempo de manera sencilla. De igual manera el conocimiento de la progresión armónica permite calcular la tonalidad y realizar una clasificación de género.

Para ello, se implementará en una primera fase el algoritmo planteado en el artículo de Bryan Pardo y William P. Birminghan, que ofrece la posibilidad de unir ambos métodos e incluso variar el orden en el que hasta ahora se afronta el problema del tempo.

3. Metodología:

Formalización del problema, y presentación teórica de
métodos usados para su resolución.

3.1. Introducción.

El tempo es un dato clave para el desarrollo de cualquier tipo de aplicación en Music Information Retrieval (MIR) ya que es la unidad básica para calcular otros descriptores simbólicos. Existen diferentes formas de afrontar la detección de tempo[10]. Este proyecto se basa en la suposición de que la duración de los paquetes armónicos es proporcional al tempo de una canción, es decir, que si dividimos 60 entre las duraciones de los paquetes armónicos que más se repiten obtendremos un valor proporcional al tempo, ya sea el doble, la mitad o el tempo. Para esto, primero se deben conocer las unidades armónicas de una canción. El problema reside en que los algoritmos que realizan un análisis armónico se basan en un tempo ya conocido, impidiendo hacer uso de ellos. La solución es realizar una segmentación de las canciones en función de la armonía con independencia de la información rítmica, de manera que cada segmento tenga coherencia armónica. El artículo "Algorithms for Chordal Analysis" de Bryan Pardo y William P. Birmingham responde a las necesidades anteriormente expuestas puesto que en él se plantea realizar la detección armónica sin necesidad de conocer las características rítmicas.

Tomando como base los conceptos planteados en dichos artículos se desarrollará un programa que detecte unidades armónicas con independencia de las características rítmicas, para posteriormente calcular el tempo, la tonalidad y clasificando su género. En una primera aproximación, el programa trabajará con ficheros MIDI para demostrar la hipótesis inicial. Tras esta primera parte se realizará una segunda aproximación que trabajará con audio real, usando como puente el programa de Antonio Pertusa y José M. Iñesta, (2008) para transformar audio real en formato wav en una secuencia MIDI.

[10] Para mas detalles ver estado de la cuestión página 16- 18

3.2. Algorithms for Chordal Analysis.

3.2.1.Introdución.

El algoritmo desarrollado por Bryan Pardo y William P. Birmingham consigue dividir una canción en unidades armónicas sin hacer uso de la información rítmica. Se puede dividir en dos partes: Segmentación y Etiquetación. Ambos puntos son complementarios, aunque se explicarán de manera independiente para facilitar su compresión.

3.2.2. Etiquetación.

La etiquetación es el proceso por el cual se asigna un tipo de acorde a un conjunto de notas. Consiste en calcular el peso de los distintos tipos de acordes y quedarnos con el de mayor peso. Para ello, se comparan las notas que existen en el trozo a analizar con las tablas que contienen los acordes con los que se trabaja. El peso es un valor entero relativo (dependiente de los pesos con los que compita) que nos da una idea de la posibilidad de que un segmento contenga un valor armónico u otro.

Para facilitar su compresión suponemos que la etiquetación se va a realizar sobre una muestra ya segmentada, por lo que ya se conoce la unidades armónicas que lo conforman. Usaremos como ejemplo la figura 1, formada por la progresión Do Mayor, Sol 7D, Do Mayor.

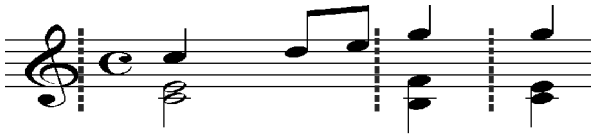


Figura 1.a. Notación musical.

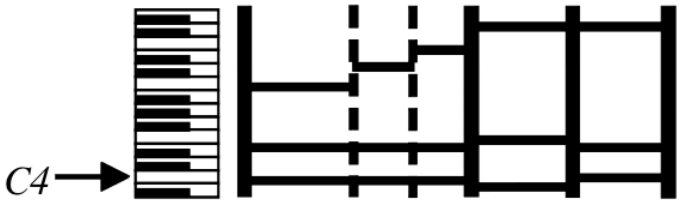


Figura 1.b. Representación pianola

La escala cromática usada en la música tonal occidental esta compuesta por doce semitonos. A cada uno le asignamos un número entero de manera progresiva sin hacer referencia a la información de octava, obteniendo la siguiente tabla:

C	C# D♭	D	D# E♭	E	F	F# G♭	G	G# A♭	A	A# B♭	B
0	1	2	3	4	5	6	7	8	9	10	11

Tabla 1: Equivalencia entre notas y números enteros.

De esta manera se puede representar cada acorde como un conjunto de números enteros, por ejemplo C Mayor será {0,4,7} y A menor {9,0,4}. Esta representación, llamada "*n-tuplas*", permite reducir el número de acordes que almacenaremos en las tablas ya que cualquier acorde se puede calcular en función de uno base (C) sumando un factor de transporte (*r)*. Solo hace falta almacenar el acorde base de C por cada tipo de acorde, el resto se calcularan fácilmente.

Por ejemplo, imaginemos que tenemos almacenado en nuestra tabla el acorde de C Mayor {0,4,7} y queremos saber cual es el acorde de G Mayor. Como G dista 7 semitonos de C, el factor de transporte es $r = 7$, al sumarlo al acorde tipo de C Mayor obtendremos {7,11,14}. Siempre que un número se salga del rango de números enteros que hemos definido (0 a 11), como el caso de 14, hay que dividir entre 12 y quedarse con el resto, en este caso 2, por lo tanto el acorde de G Mayor calculado será {7,11,2} que traducido a notas es {G, B, D}, que efectivamente, forman el acorde G Mayor. Como ya se ha dicho anteriormente, sólo es necesario almacenar un acorde por cada tipo, ya que el resto se calcula de manera sencilla sumando r al acorde base. Es fácil de deducir que el valor de r coincide con el valor entero de la tónica del acorde, ya que coincide con el número de semitonos que dista de C, lo que simplifica el cálculo.

El número y tipo de acordes que se usen depende del tipo de música a analizar y del problema a abordar. Para esta investigación, se usarán 7 tipos de acordes distintos: perfecto mayor (PM), perfecto menor (pm), séptima de dominante (7 Dom), 4º suspendida (4 sus), 2º suspendida (2 sus), aumentado (Aum) y disminuido (Dis), que son los más comunes en la música clásica hasta el inicio del romanticismo en los principios del siglo XIX y música popular. Las tablas de estos acordes referidas a C son:

PM	*pm*	*7 Dom*	*4 sus*	*2 sus*	*Aum*	*Dis*
{0,4,7}	*{0,3,7}*	*{0,4,7,10}*	*{0,5,7}*	*{0,2,7}*	*{0,4,8}*	*{0,3,6}*

Antes de describir el procedimiento a seguir para calcular los pesos y el nombre del acorde, se deben conocer algunos conceptos básicos que van a ser usados:

Los cambios armónicos sólo pueden producirse cuando una nota comienza o termina. Se define un *punto de separación* (*partition point*) siempre que ocurra un nuevo evento de inicio o final de nota, o lo que es lo mismo cuando un conjunto de notas que están sonando a la vez cambia, ya sea porque una nota deja de sonar o porque se añade una nueva. De manera igualmente sencilla se puede definir un *segmento (segment)* como un intervalo continuo entre dos *puntos de separación* cualesquiera. Por último un *segmento mínimo (minimal segment)* es un intervalo entre dos puntos de separación correlativos. La figura 2 muestra estos conceptos aplicados al ejemplo de las figuras 1.a. y 1.b .

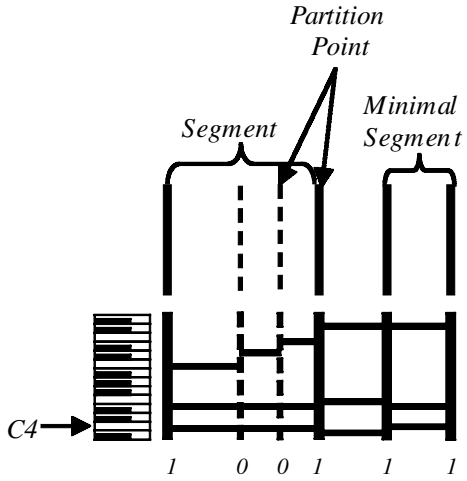


Figura 2.

Para determinar el acorde presente en cada segmento, se calculará el peso de todos los acordes de cada tipo, y se asignará el de mayor peso. Para calcular el peso de un acorde, se aplica la formula : S = P - (M+N), donde P se define como el número de notas del acorde a comprobar que están presentes en el segmento analizado, M como el número

de notas presentes en el acorde que no están en el segmento tratado y por último N como el número de notas presentes en el segmento que no están en el acorde.

Para ilustrar el procedimiento se usará como ejemplo el primer segmento (formado por tres segmentos mínimos) de la figura 1, que contiene las notas: C, E, D. Para el cálculo de los parámetros que conforman la fórmula es necesario conocer el número de veces que esta sonando una nota que se identifica con el número de *segmentos mínimos* que abarca. De esta manera la nota C tendrá un valor de 4 puesto que el C4 abarca 3 segmentos mínimos y C5 sólo 1. De igual manera se calcula el valor de E, que da como resultado 4, y el de D, que vale 1. Por tanto el peso total del acorde C mayor para este segmento es:

$$S = P - (M + N) = (\text{Peso}(C) + \text{Peso}(E)) - \text{Peso}(G) - \text{Peso}(D) = (4 + 4) - 1 - 1 = 6$$

Si se calcula el peso de este segmento con D menor {D, F, A} tendremos que P = Peso(D) = 1, puesto que de las notas que conforman el acorde de D menor sólo D está presente en el segmento; M = 2 ya que ni F ni A están en el segmento y por último N = Peso(C)+ Peso(E) = 8. El peso final S es -9. Los resultados obtenidos con este método son coherentes con la intuición que nos dice que el primer segmento del ejemplo 1 puede corresponder a C Mayor pero nunca a D menor. Se repite el proceso con todos los acordes de todos los tipos almacenados en las tablas. El tipo de acorde que se asigna a cada segmento corresponde al de mayor peso.

En caso de que dos acordes tengan el mismo peso existen varias fórmulas de desempate: seleccionar el acorde cuyo tipo sea más frecuente; si uno de los acordes implicados en el empate es el mismo que el calculado anteriormente, optar por repetir la estructura armónica, o todo lo contrario (variar la progresión armónica para darle mas variedad: elegir el acorde que tenga la tónica con mayor peso). Dado que el objetivo final de este proyecto no es tanto calcular de manera exacta el tipo de acorde como saber que ese segmento forma una unidad armónica. En nuestro caso, siempre que existan dos acordes que tengan el mismo peso nos quedaremos con el tipo de acorde más frecuente.

3.1.2.3. Segmentación.

Segmentar, dividir en unidades mas pequeñas, se puede realizar en función de muchos parámetros distintos: unidades rítmicas, unidades melódicas, frases/ respuestas, etc. El algoritmo de Bryan Pardo y William P. Birmingham toma como base de segmentación la armonía.

El proceso de análisis armónico es complejo ya que por regla general no se exponen los acordes de manera aislada, si no que se entremezclan melodías, contrapuntos, juegos rítmicos... Hecho que se ve agravado por la evolución que ha sufrido la música a lo largo de los siglos en todos los campos y especialmente en el armónico, lo que ha supuesto una búsqueda de nuevos espacios sonoros que no se conforman con los acordes tonales tradicionales. Por tanto, nos encontramos con dos problemas: la dificultad para detectar de manera clara conjuntos armónicos, ya que no siguen ningún tipo de regla (solo la creatividad del compositor), lo que hace muy difícil su análisis computacional; y la gran variedad armónica que tenemos hoy en día sobre todo en la música culta y las vanguardias.

Por tanto, este proyecto no pretende realizar una segmentación perfecta capaz de analizar cualquier tipo de música, máxime si muchos pasajes presentan diferentes interpretaciones en función del músico que la analice. Lo que se busca es obtener los mejores resultados en el mayor número de canciones de distintos estilos, conocer las limitaciones del programa pero sobre todo demostrar que la armonía es el camino correcto para calcular el tempo: suponiendo que los paquetes armónicos tienen relación directa con el tempo y calculándolo de manera más segura y fiable. Además de esto, el análisis armónico se usará para calcular la tonalidad y clasificar género de cada canción.

Un cambio armónico sólo puede producirse cuando un conjunto de notas cambia. Esto puede suceder por dos motivos: o alguna de las notas que están sonando deja de hacerlo o a dicho conjunto se le añade alguna nota nueva. La traducción de esto al programa es que un cambio armónico solo puede ocurrir en los *puntos de separación*. La segmentación asigna a cada uno de los puntos de separación un valor 1 en el caso de que se de un cambio armónico o 0 cuando no se exista ninguno. De esta manera la segmentación se puede entender como un array binario de longitud p (nº de puntos de separación) con el inicio y el final siempre a 1 y donde existen 2^{p-2} posibilidades distintas de segmentación. El

número de puntos de separación está limitado como máximo al doble del número de notas (esto sólo se da en las canciones donde ningún par de notas coinciden en su inicio o final), por lo que el número máximo de *puntos de separación* es igual a 2*n*, siendo *n* el número de notas. Una canción tiene $2^{p-2} = 2^{2n-2}$ posibilidades distintas de en las que segmentarse. El proceso de segmentación se elegirá la que maximice el resultado.

Si almacenamos todos los segmentos de una canción (la música es secuencial por lo que no tiene sentido crear segmentos con notas anteriores) y calculamos sus respectivos pesos se obtendrá algo parecido a la tabla siguiente, referida al ejemplo de la figura 1:

	1	*2*	*3*	*4*	*5*
0	*C Mayor - 2*	*C Mayor - 3*	*C Mayor - 6*	*C Mayor - 6*	*C Mayor - 9*
1		*C Mayor - 0*	*C Mayor - 3*	*C Mayor - 3*	*C Mayor - 6*
2			*C Mayor - 2*	*C Mayor - 2*	*C Mayor - 5*
3				*G7 - 2*	*G7 - 2*
4					*C Mayor - 3*

Tabla 1

Las filas corresponden al inicio de cada punto de separación mientras que las columnas al final. Cada casilla representa el segmento formado por la fila y la columna que le corresponde, en ellas se almacena el acorde detectado y su peso usando el método del apartado 3.1.2. Por ejemplo la casilla 1-3 almacena el peso y el acorde que corresponde al segmento que comienza en el segundo punto de separación y termina en el cuarto. Si analizamos la tabla anterior. Se llega a la conclusión de que el problema se puede afrontar como un grafo, en concreto un grafo acíclico dirigido (DAG) ya que cumple todas las características (ninguno de los vértices tiene un camino directo que empiece y termine en ellos, y todos los caminos son dirigidos). Un DAG es un grafo que no tiene ciclos, es decir que uno existe ningún camino directo que empiece y termine en un vértice n. Al ser dirigido, cada camino *n* tiene una dirección determinada, se inicia en un vértice A y termina en un vértice B.

El siguiente paso para la segmentación es construir el grafo con toda la información de una canción, en donde cada uno de los puntos de separación encontrados son los vértices y los caminos son las uniones de cada vértice con sus posteriores. El valor de cada camino es el peso del segmento delimitado por el vértice de inicio y el vértice final de cada camino, por ejemplo un camino que vaya del último vértice al primero tendrá como valor el peso del segmento que engloba toda la canción, mientras que uno que vaya del tercer vértice al quinto tendrá como valor el peso del segmento que forma entre ellos. En la figura 3 se puede ver el grafo de la figura 1 del apartado anterior.

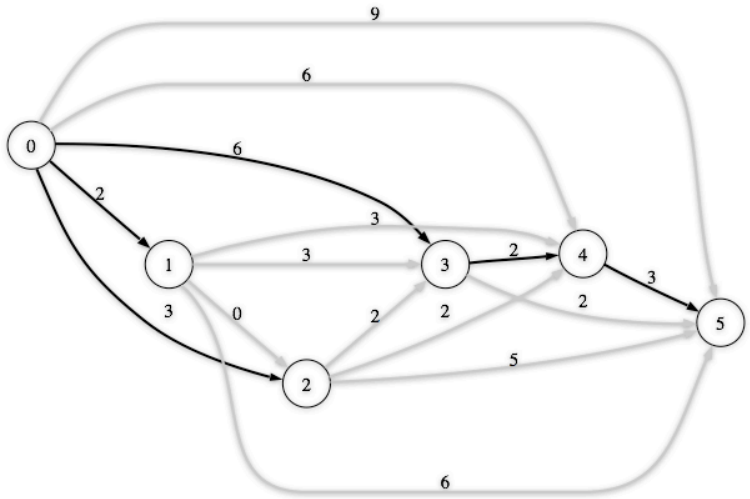


Figura 3

Como se puede comprobar en la figura 3 siempre existe un camino entre el vértice *n* y sus posteriores pero nunca con sus anteriores. Por último se halla el camino mas largo que define los grupos armónicos en los que se divide la canción. El camino mas largo es la unión de los segmentos necesarios para ir desde el vértice de inicio al vértice final con el mayor peso. El peso es un indicador relativo (puesto que su valor es más o menos relevante en función de los que le rodean) de la posibilidad de una unidad armónica.

Una vez determinado el camino más largo, los vértices que formen parte de él serán los puntos de separación en donde existirá un cambio armónico y la unión de cada uno de ellos indica el tipo de acorde que forma cada unidad. Si nos fijamos en la tabla 1 y en la figura 3 podemos determinar fácilmente el resultado final:

Camino	*1º*	*2º*	*3º*
Vértices	*0-3*	*3-4*	*4-5*
Acorde del camino	*C Mayor*	*G 7*	*C Mayor*

Este resultado es el mismo que obtendríamos si se realizara un análisis manual del pasaje de la figura 1.

3.3. Tempo.

El tempo representa la velocidad de ejecución de una pieza musical y determina la duración de cada nota. Se define como el número de pulsos por minuto. El pulso es el latido de la música, el chasquido de dedos que todos realizamos al escuchar una canción. La duración de un pulso en una canción en función del tempo es: $d_p = 60/t$.

Una vez tenemos segmentada la canción en unidades armónicas gracias al proceso planteado por Bryan Pardo y William P. Birmingham (2002), se calculará la duración de cada uno de los paquetes armónicos y se formará un historial de todos ellos, se permite un margen de error de *e* segundos, es decir, si una unidad armónica tiene de duración *d* y otra $d \pm e$ se tomará como la misma duración, se seleccionará la duración que más se repita. Para obtener el tempo se aplica $t = 60/d_a$, donde d_a es la duración armónica que más se repite.

La relación de $t = 60/d_a$, se deduce de manera sencilla. Si suponemos que la duración armónica que más se repite coincide el pulso de la canción $d_p = d_a$, por lo que $d_a = 60/t$ de donde despejamos el tempo $tempo = 60/d_a$.

3.4. Beat.

El beat índica donde se producen los golpes del tempos dentro de la canción. Sirve para ajustar las notas de la canción en los tempos. Una vez calculado el tempo general se buscarán los beats donde se producen los onsets en base al tempo calculado, usando un margen de desviación de *e*. Una vez detectados todos los beats se realizará un histograma y se volverá a calcular el tempo en base a la distancia entre onsets (IOI[11]) mas frecuente.

3.5. Tonalidad.

Una vez calculada la progresión armónica de una canción se calculará su tonalidad, para ello se buscará la cadencia perfecta (V-I) que más se repite. Se realizará un análisis de todas las cadencias V-I que se dan en la canción y se almacenará su duración. La tonalidad estará definida por la cadencia perfecta que mas tiempo este presente en la canción. Para comprobar si un par de acordes correlativos forman una cadencia perfecta usaremos la tabla de valores enteros definida para el cálculo de los pesos:

C	C# D♭	D	D# E♭	E	F	F# G♭	G	G# A♭	A	A# B♭	B
0	1	2	3	4	5	6	7	8	9	10	11

a cada acorde se le asignará como valor el peso que le corresponde a su raíz. Este pero no tiene nada que ver con el usado en el cálculo del peso de cada acorde, es usado para calcular la relación interválica entre cada par de acordes que se calculla como la resta entre el valor del primer acorde (*v1*) y el valor del segundo (*v2*). Si el resultado de esta resta es 7 o -5 se trata de una cadencia perfecta puesto que el primer acorde es el quinto grado del segundo. El valor 7 se da cuando el I grado sea un acorde comprendido entre C y E mientras que -5 para el resto, de F a B.

[11] Inter-Onset-Interval

3.6. Clasificación de género.

La clasificación de género es un punto muy importante en la organización de bases de datos musicales. Usamos estas clasificaciones en nuestro día a día, por ejemplo al entrar a una tienda de música, recomendar una canción o organizar nuestra colección de discos. La clasificación automática de género ha cobrado gran importancia dentro de las investigaciones desarrolladas en MIR ya que permite agilizar el proceso, sobre todo cuando las bases de datos son muy grandes.

El sistema de clasificación automática de género de Carlos Pérez-Sancho, David Rizo y José M. Iñesta (2008), usa las progresiones armónicas de una canción para clasificarla en un género u otro, usando para ello una base de datos obtenida tras entrenamiento previo.

La longitud de las progresiones armónicas en las que se dividirá una canción para su análisis se mide en n-gramas. El valor de n indica el número de acordes que se tomará en cada progresión de manera que si se habla de 2-gramas se usarán progresiones formadas por dos acordes, 3-gramas, 3 acordes y así sucesivamente. El inicio de cada progresión siempre es el último acorde de la progresión anterior. En el ejemplo siguiente se muestra como se dividiría una progresión armónica en función de n-gramas de valor 2 (bigramas) y 3 (trigramas).

Progresión:	I - V - VI - IV - V - I ;				
Bigrama:	I - V ;	V - VI ;	VI - IV ;	IV - V ;	V - I ;
Trigrama	I - V - VI ;	V - VI - IV ;	VI - IV - V ;	IV - V - I ;	

Para el entrenamiento se usan 9 de las diez partes en las que se divide las canciones con las que se va a experimentar. Consiste en usar canciones con el género ya etiquetado previamente para generar secuencias de progresiones armónicas de n-gramas que definen cada género. Una vez obtenidos los n-gramas de cada género se clasificará el resto de canciones comparando sus progresiones armónicas con las progresiones generales de cada género utilizando modelos estocásticos del lenguaje. Este proceso se repetirá para todas y cada una de las diez partes en las que se divide la base de datos, obteniendo como resultado final la media de los resultados individuales.

3.7. División en fragmentos.

Una vez implementado el algoritmo de Bryan Pardo y William P. Birmingham comienza la experimentación. Al probar con las primeras canciones se comprueba que el tiempo de ejecución es muy grande, del orden de 14 minutos por canción. El problema reside en el cálculo del grafo y en la resolución del camino más largo. Se comprueba el número de veces que se calcula el peso en la pieza Bourre en Dm de Handel con 5139 puntos de separación: el grafo tendrá una tabla asociada (coma la Tabla 1) de 5139x5139 elementos, donde todos los elementos que estén por debajo de la diagonal principal valen 0. Por cada casilla del grafo, se formará un segmento y se calculará su peso, lo que implica probar con 11 acordes distintos por cada tipo (7 en nuestro caso). Por tanto tendremos que el número de veces que se realiza el cálculo de un peso es de 7x11x (la suma de todos los segmentos que se puedan formar) en nuestro ejemplo:

$$\sum_{n=1}^{n=5139} 7 \cdot 11 \cdot (n\text{-}1)$$

El número veces que se realiza la operación S = P - (M+N) es muy elevado, lo que hace al programa muy lento. Un análisis pormenorizado de todos y cada uno de los segmentos de manera general es poco práctico y tedioso. Sin embargo, si suponemos que el análisis de un segmento depende de los que les rodea y no de los que dista mucho, podemos fragmentar la canción en subconjuntos de manera que cada trozo pueda ser analizado de manera independiente[12]. Por tanto una canción se analizará partiéndola en diferentes subconjuntos de tamaño *k* que serán analizados de manera independiente como si se tratase de canciones de duración más corta. Para la creación de cada nuevo subconjunto se tomará como inicio el instante correspondiente al penúltimo punto donde se ha detectado un cambio de armonía en el segmento anterior.

[12] Los resultados de esta suposición se demuestran en la parte de experimentación en la página 40 - 43.

3.8. Selección de acordes.

Un punto muy importante es la selección de los acordes base que se van a incluir en el programa para realizar la segmentación armónica, ya que esta selección condiciona el resultado de los análisis en función del estilo de música. No es lo mismo realizar un análisis armónico de una canción del clasicismo que de una pieza de jazz.

Los acordes que se han tomado como base son: el acorde perfecto mayor, perfecto menor, 7 de dominante, 2º suspendida, 4 º suspendida, aumentados y disminuidos. Estos acordes son los mas usados hasta el romanticismo (inicios del siglo XIX), sobre todo en el clasicismo en donde se tenía muy en cuenta las reglas amónicas y no tanto en los periodos anteriores, donde no existía una teoría armónica como tal que definiera la jerarquía de un acorde en función de su grado (aunque en la práctica si que se usaban las nociones básicas producto más de una superposición de voces en busca de una sonoridad concreta que de un estudio de las posibilidades armónicas). Con esta selección se asegura un buen resultado con canciones de música clásica occidental hasta el romanticismo[13], y a su vez nos permite afrontar la música pop y rock con garantías, puesto que los acordes perfecto mayor, perfecto menor y séptima de dominante son los mas usados. Por contra, a la hora de afrontar el jazz, por ejemplo, se deberá cambiar de acordes base, puesto que los elegidos no están casi presentes en este tipo de música.

Los estilos más difíciles de analizar son el jazz y la música contemporánea. En el jazz no existen unos acordes básicos que lo definan, ya que tiene una componente de improvisación muy grande y la base armónica puede variar mucho en función de la calidad de los instrumentistas. En la música contemporánea, el concepto de armonía desaparece como tal y no se puede afrontar desde un óptica clásica.

[13] Ver los resultados en el apartado 5 de experimentación, página 43-53.

3.9. Transcripción audio - midi.

3.9.1. Introducción.

Una vez desarrollado el algoritmo capaz de calcular el tempo en función de la armonía, la tonalidad y la clasificación de género usando ficheros MIDI se analizará el audio real. Para ello usaremos el sistema de transcripción de Pertusa y Iñesta (2008) que transforma la información de audio real de una una canción en una secuencia MIDI.

Este trabajo se basa en el hecho de que la envolvente de un sonido real tiende a variar suavemente en función de la frecuencia, y propone que es posible detectar cada frecuencia fundamental haciendo un análisis de los todos los armónicos presentes.

3.9.2. Metodología.

El sistema de transcripción polifónica convierte una señal de audio mono en una secuencia de notas MIDI. La canción se separa en cuadros usando una ventana Hanning de 93 ms (4096 muestras para una señal muestreada a 44.1 kHz) que avanza en intervalos de 46 ms. A cada cuadro se le aplica la transformada de Fourier, realizando así una STFT (Short Time Fourier Transform). El algoritmo trabaja cuadro a cuadro y el proceso es idéntico para todos ellos.

Una vez que se tiene el espectro de cada cuadro se buscan los picos espectrales cuya amplitud supere un umbral p y se desecha todos los demás. Estos picos son los posibles candidatos a frecuencia fundamental que están sonando. Se debe cumplir dos requisitos más para ser un candidato: que se encuentre dentro de un margen $[f_{min}, f_{max}]$ establecido. Para encontrar los armónicos se usará $hf_0 \pm f_\tau$ donde $h = 2, 3, 4...$ y f_τ un margen constante alrededor de donde debería estar el armónico, para considerar desviaciones espectrales.

Una vez que todos los candidatos han sido seleccionados, se ordenan de forma decreciente en función de la suma de las amplitudes de sus armónicos. De entre todos los candidatos detectados se usarán como con los F primeros, el resto son desechados. A continuación se calculan todas las combinaciones que pueden formar todos los candidatos seleccionados y se seleccionará la de mayor peso. Esta combinación es la que define las notas que estarán presentes en el cuadro analizado.

El peso de una combinación se define como la suma del peso individual de todos los candidatos. Para calcular el peso de cada candidato *c* es necesario conocer su estructura espectral p (el valor de sus parciales armónicos $hf_0 \pm f_\tau$ donde h = 2, 3, 4, ... ,10) que forma un vector de amplitudes $\mathbf{p_c}$:

$$\mathbf{p}_c = \{p_{c,1}, p_{c,2}, \ldots, p_{c,h}, \ldots, p_{c,H}\}$$

Un mismo parcial puede estar compartido por mas de un candidato, o un parcial que para una combinación solo pertenece a un candidato, en otra puede estar presente en varios, por lo que el cálculo del vector p_c no es inmediato. Por esto se debe realizar una clasificación de los parciales armónicos que componen la combinación que se está tratando en armónicos compartidos y armónicos no compartidos.

Para el calculo del vector $\mathbf{p_c}$ de armónicos de un candidato se analizan los armónicos que lo componen, los del tipo no compartidos pasan directamente al vector de amplitudes p_c. Para saber qué parte de amplitud correspondiente a los armónicos compartidos pertenece al candidato analizado se realiza una interpolación con los armónicos no compartidos. Si la interpolación es mayor que el armónico compartido todo valor de dicho armónico pertenece al candidato que se está analizando y no tiene ningún aporte para el otros candidatos. En caso de que la amplitud de la interpolación en el parcial armónico compartido sea menor que dicho parcial, se asignará el tanto por ciento que este por debajo de la interpolación al vector $\mathbf{p_c}$ y el resto al parcial armónico compartido (tanto por ciento que queda por encima de la interpolación). De esta manera se calcula el valor de todos los parciales armónicos de los candidatos de la combinación analizada separando en los parciales compartidos, el valor de cada candidato.

Una vez que se conocen todos los vector $\mathbf{p_c}$ de los candidatos de la combinación a tratar se calcula el peso de cada candidato teniendo en cuenta la suma de las amplitudes armónicas y la suavidad espectral. El peso de una combinación es la suma del peso de cada candidato. La notas que se extraen en el cuadro tratado son las que forman la combinación con mayor peso.

3.10. Resumen.

A la hora de afrontar la segmentación de una canción en función de su armonía siempre se seguirá el mismo proceso:

1º Seleccionar el primer trozo, de 0 a *k* segundos.

2º Determinar todos los puntos de separación del trozo a analizar.

3º Formar el grafo creando tantos nodos como puntos de separación existan. Los caminos que relacionan cada nodo con sus posteriores y el valor de estos que es el peso del segmento que forma con cada uno de ellos.

4º Búsqueda del camino más largo en el grafo y asignación de los puntos de separación que forman inicio y final de una unidad armónica.

5º Cálculo de las duraciones de todas las unidades armónicas y creación de un histograma con todas ellas teniendo en cuenta un margen de desviación *e*.

5º Calculo del tempo relativo del segmento: para ello nos quedamos con la duración armónica (d_a) que mas se repita y dividimos $60/d_a$.

6º Selección del siguiente tramo a analizar, tendrá como inicio el penúltimo cambio armónico del segmento anterior, y como duración *k*.

7º Vuelta al paso dos hasta analizar todos los segmentos.

8º Cálculo del tempo general de la canción teniendo en cuenta todos los cambios armónicos. Este cálculo es igual que el realizado en el apartado cinco.

9º Cálculo del beat en función del tempo general.

10º Cálculo de la tonalidad buscando la cadencia perfecta (V-I) que se repite durante más tiempo.

11º Clasificación de género una vez que todas las canciones hayan sido analizadas.

4. Código[14]

4.1. Introducción.

El programa se puede dividir en dos partes bien diferenciadas: una primera que se encarga de recoger la información del fichero de entrada y procesarla, transformándola en una representación pianola; y una segunda que realiza la segmentación de la canción en función de la armonía y la tonalidad:

1º Lectura de la información de entrada.

2º Transformación en una pianola.

3º Análisis de cada cuadro:

a) Segmentación

b) Cálculo del histograma de duraciones armónicas.

c) Cálculo del tempo.

4º Una vez que se han analizado todos los cuadros se calcula el tempo general:

a) Calculo del histograma de duraciones armónicas.

b) Cálculo del tempo general.

5º Cálculo del beat.

6º Calculo de la tonalidad.

El resultado son tres ficheros de texto. En uno se almacenarán los instantes de tiempo donde se ha detectado un cambio de armonía y el valor del mismo (tipo de acorde del que se trata). En otro tendremos los tempos individuales de cada segmento y el tempo total de toda la canción. El último fichero corresponde a los instantes en donde se da un beat.

---

[14] El código completo se adjunta en los apéndices , página 65 - 86.

4.2: Organización y lectura de la información.

En primera instancia el programa se encarga de leer la información del fichero de entrada y transformarla para poder trabajar con ella. Tras esto se obtiene una matriz de 129 columnas y tantas filas como eventos se han producido. En la primera columna se almacena el vector de tiempos que indica cuando se produce un evento. Las 128 restantes corresponden a las notas que están sonando en cada instante de tiempo.

El programa trabaja con ficheros de texto. Para transformar un fichero MIDI en un fichero tipo texto usaremos el algoritmo smf2txt[15] desarrollado por el grupo de trabajo "Computer Music Laboratory" de la Universidad de Alicante. Este programa devuelve un fichero de texto con la estructura tiempo de inicio de un evento, duración y altura o pitch. El programa permite variar el número de analizadas así como la estructura del fichero resultante. Para la experimentación, se eliminará la cabecera del fichero de salida y la pista de percusión.

La información que genera smf2txt está en función de los eventos producidos, una línea por evento. Es necesario transformar la información para que el tiempo sea el aspecto más importante, en una línea todos los eventos que se producen a la vez:

1) Se calcularán todos los eventos que se producen en el fichero, los eventos nota_on son inmediatos mientras que los nota_off son la suma de nota_on mas duración.

2) Se ordenarán todos los eventos que se producen a la vez, consiguiendo un vector de tiempos que lleva asociado otro de 127 elementos, donde se almacenarán los eventos que se producen en cada instante con un 1 notas activas y 0 no activas.

Una vez que tenemos la información en representación pianola pasamos al análisis de cada cuadro de duración *k*.

[15] disponible en http://grfia.dlsi.ua.es/gen.php?id=resources

4.2: Segmentación.

En esta parte se fragmenta cada cuadro de duración *k* en unidades armónicas sin necesidad de conocer el tempo, métrica, tonalidad, u otros conceptos musicales, simplemente se analiza lo que esta sonando. El programa se basa en el artículo de Bryan Pardo y William P. Birminghan (2002)[16]. Una vez segmentada la canción usaremos esta información para calcular el tempo y la tonalidad. Por último, una vez que todas las canciones hayan sido analizadas se procederá a la clasificación de género.

Tal y como se describe en la metodología, se formará un grafo de tantos vértices como puntos de separación existan, y caminos que unen un vértice con todos sus posteriores, el valor del cada camino es el peso del segmento que forman. Para calcular el peso se usará la formula Peso = P - (M+N) donde P representa el número de veces que están presentes las notas del acorde en el segmento analizado, M las notas del acorde que no aparecen en el segmento y por último N, el número de notas ajenas al acorde que aparecen en el segmento. El análisis de cada cuadro es siempre el mismo:

1) Formación del cuadro que se va a analizar. El inicio es siempre el penúltimo punto donde se ha calculado un cambio armónico del cuadro anterior y la duración constante e igual a *k*, a excepción del primer cuadro que empieza en 0 y el último, que puede tener una duración menor que *k*.

2)Análisis de cada cuadro. En caso de que no se produzca ningún cambio armónico se repetirá el proceso aumentando la longitud del segmento al doble hasta que se detecte alguno. La variación de la longitud del segmento solo afecta a estos casos particulares.

2) Se crea la tabla que contiene todos los valores necesarios para el grafo: número de puntos de separación = número de nodos. Los caminos son los segmentos formados por la unión de un vértice con todos sus posteriores. Para cada camino se calcula el peso y se le asocia un tipo de acorde.

[16] Ver apartado 3.2. Algorithms for Chordal Analysis en la página 20 - 27.

3) Análisis del grafo (DAG) y cálculo del camino más largo. Para este cálculo se hará uso de la función graph.h que devuelve un vector de enteros con los vértices que forman el camino más largo. A este vector hay que añadirle el primer y último vértice que también forman parte del camino.

4) Se traduce la información del camino más largo a tiempos y cambios armónicos haciendo uso de la tabla construida anteriormente, que contiene todas las combinaciones de caminos, el peso que le corresponde a cada uno y el tipo de acorde. Sólo hace falta buscar qué caminos están en la ruta del camino más largo.

7) Cálculo del tempo: primero se crea un histograma con margen de error de *e* segundos, donde se almacenan todas las duraciones de los paquetes armónicos. La duración de cada paquete armónico es muy sencilla de calcular ya que consiste en restar el tiempo final menos el tiempo de inicio del paquete. Cada instante donde se da un cambio armónico corresponde al inicio de un paquete y el final de otro, excepto el primero y el último, que se corresponde con el inicio del primer paquete y el final del último, respectivamente. Tras esto nos quedamos con el valor de la duración que más ser repite, d_a. Por último se calcula el tempo aplicando la fórmula siguiente[17]. Tempo = 60/*d* d_a.

6) Escritura de la información en los ficheros de texto en donde se almacenarán los tiempos de los cambios armónicos y la progresión. En todo momento se debe tener conocimiento de la canción en su conjunto, para ello se almacenarán los cambios armónicos y la progresión en variables que puedan ser usadas por el resto del programa. Vuelta al inicio.

8) Una vez que se han analizado todos los paquetes de manera individual, se calculará el tempo de la canción y el beat. Para ello se repite el punto número 7 con los cambios armónicos de la canción.

[17] Ver apartado 3.3. Tempo en la página 27 donde se encuentra la explicación.

9) Cálculo de la tonalidad: una vez que conocemos la progresión armónica de la canción buscaremos todos las cadencias perfectas mayores (V-I) que existan. A la hora de llevar la cuenta de cada una se almacena la duración de la misma. La tonalidad corresponderá al I grado de la cadencia perfecto mayor que más se repita.

4.3: Aspectos a tener en cuenta.

Para calcular el peso final de cada segmento, primero se calcula el peso para todos los acordes por cada tipo, quedándonos con el de mayor peso. Esto supone un coste computacional muy grande puesto que por cada segmento a analizar hay que calcular 11 pesos por cada tipo de acorde, que como mínimo deben de ser tres tipos: Mayores, menores y 7 de dominante, aunque el tipo y el número de acordes depende del tipo de música que se vaya a analizar. Por este motivo se ha introducido el factor de fragmentación *k* que puede tener valores sobre 5 segundos sin que eso altere el resultado, aunque siempre es recomendable usar valores un poco más grandes del orden de 10 segundos.

Cuando se realiza el paso de MIDI a texto se debe eliminar la pista de percusión que por regla general suele estar compuesta por instrumentos de altura indeterminada y que no se corresponden a ninguna nota en concreto sino a un sonido de percusivo, por lo que si se utilizan en el análisis puede suponer un serio problema añadiendo notas que no tienen que ver con la armonía general.

Muchos MIDIs no están cuantificados, lo que hace que eventos que se tendrían que producir en el mismo instante de tiempo se produzcan en instantes diferentes. Normalmente esta diferencia es muy pequeña, del orden de 0.002 segundos, pero que traducidos al algoritmo implica nuevos puntos de separación que hacen que el cálculo sea mayor, aumentando el tiempo de ejecución del programa. También es una gran fuente de errores puesto que en muchos casos se detectan cambios armónicos en puntos que no existen en notación musical. La solución es que a la hora de transformar el fichero de entrada en pianola se agrupen todos los eventos que deberían producirse a la vez. El programa permite al usuario elegir entre cuantificar o no, también existe la opción de incluir el valor de cuantificación.

5. Experimentación.

Validación del programa realizando diferentes experimentos
que demuestren su utilidad.

5.1. Introducción.

En este apartado se detallarán los experimentos realizados para validar el programa desde diferentes puntos de vista: cálculo del tempo, detección de tonalidad y clasificación de géneros musicales. Por contra, no se hará referencia a los experimentos usados para depurar el programa, a excepción del experimento que demuestra que una canción puede ser entendida como un subconjunto de otra.

Se han usado más de 850 canciones agrupadas en tres estilos: Académica, Jazz, Popular. Cada uno a su vez tiene 3 sub-estilos, de manera que dentro de la música académica tendremos: Barroco (56 canciones), Clasicismo (50 canciones), Romanticismo (129 canciones), en jazz: Bop (94 canciones), Bossanova (66 canciones), Pre (178 canciones) y por último en popular: Blues (84 canciones), Cetic (99 canciones) y Pop (100 canciones). Los experimentos están referidos a dicha base de datos.

A la hora de comprobar el funcionamiento de programa sobre audio real se ha sintetizado la base de datos anterior a audio y posteriormente de nuevo a MIDI usando el sistema de transcripción de Pertusa e Iñesta (2008). Todos los experimentos constarán de dos partes: Una primera donde se detallara el funcionamiento general sobre MIDI con sus correspondientes resultados y una segunda donde se expondrán los resultados obtenidos para audio real transcrito.

5.2. División en segmentos.

En primera instancia se comprobó el tiempo de ejecución del programa sin realizar ningún tipo de segmentación sobre una base aleatoria de canciones (3 por cada subgénero de música académica) y los tiempos de ejecución obtenidos son los siguientes:

Nombre de la canción	Tiempo de ejecución
Bourree In Dm Fireworks Music, Handel	41 minutos 44,701 segundos
The Four Seasons Spring: 1st Mvt, Vivaldi	3 minutos 16.632 segundos
La Rejouissance Fireworks Music, Handel.	16 minutos 31.470 segundos
Sonata In F: 1st Mvt, Haydn	35 minutos 28.942 segundos
Sonata 16 1st Mvt, Mozart	24 minutos 28.545 segundos
Sonata In Gm op.49 no1, Beethoven	26 minutos 55.607 segundos
Tannhauser March, Wagner	6 minutos 31.562 segundos
Valse In Cm Op.64 no2, Chopin	35 minutos 4.562 segundos
Waltz 1 In B Op.39, Brahms	33 minutos 8.226 segundos

Tabla 1: tiempo de ejecución.

Como muestra la tabla anterior, el tiempo de ejecución del programa al analizar una canción completa es muy grande, por lo que es necesario agilizar el proceso de alguna manera. El problema está en el cálculo de los pesos, ya que debido a la forma que tiene el grafo supone un coste computacional de :

$$\sum_{n=1}^{n=k} a \cdot 11 \cdot (n-1)$$

Donde k indica el número de puntos de separación que existen en la canción y a el número de acordes con los que se trabaje.

Para reducir el tiempo de ejecución existen dos alternativas: reducir el número de acordes (a) o reducir el número de puntos de separación (k). Es mucho mas interesante reducir el número de puntos de separación a analizar, puesto que se está reduciendo el grado del sumatorio, mientras que disminuyendo a solo se varía una constante. Esto sólo se

consigue trabajando con unidades temporales menores que la canción original para reducir el número de puntos de separación de cada tramo.

Analizar la canción por segmentos plantea la duda de si los resultados (detección de un cambio armónico) serán correctos o no, puesto que en el artículo de Bryan Pardo y William P. Birminghan (2002) plantea que los cambios armónicos dependen de todos los elementos. Se ha probado con más de 10 y en todas ellas los resultados muestran cambios armónicos son constantes e independientes de la constante de fragmentación, *k*, lo que hace que cada segmento puede ser analizado de forma independiente, como sí de una canción distintas se trata. Esto permite afirmar que los cambios armónicos de una canción depende de los acordes que les son próximos pero poco de los que dista mucho. La tabla 1 muestra la evolución del tiempo de ejecución en función de *k* para la canción Bourree In Dm Fireworks Music de Handel.

k	Tiempo de ejecución
5	0m 34.087s
7	0m 49.634s
10	1m14.963s
15	1m 58.565s
20	2m 40.979s
25	3m 35.339s
30	4m 23.401s
60	9m 37.289s

Tabla 2: tiempo de ejecución para distintos valores de k.

Como se puede ver, la diferencia es considerable. Por tanto para el resto de experimentos se usará una constante de fragmentación igual a 10 que asegura segmentos los suficientemente grandes como para poder detectar cambios armónicos en ellos y un tiempo de ejecución optimo. En caso de no encontrarse ningún cambio armónico dentro de un segmento, se repetirá el análisis tantas veces como sea necesario hasta encontrar como mínimo un cambio armónico[18], aumentando, en cada una de las iteraciones, el valor de *k* al doble. La canción se fragmente únicamente en donde existe un punto de separación, por lo que aunque el valor de *k* es constante, la duración de cada cuadro varía en función de donde se produzca el último punto de separación que entra dentro del cuadro.

[18] El inicio y el final del segmento no se tiene en cuenta.

Por último comprobaremos como el tiempo de ejecución se reduce de manera considerable al usar una constante de fragmentación:

Nombre de la canción	Tiempo de ejecución con k =10
Bourree In Dm Fireworks Music, Handel	1 minuto 14.963 segundos
The Four Seasons Spring: 1st Mvt, Vivaldi	0 minutos 15.377 segundos
La Rejouissance Fireworks Music, Handel.	0 minutos 41.658 segundos
Sonata In F: 1st Mvt, Haydn	1 minuto 20.435 segundos
Sonata 16 1st Mvt, Mozart	0 minutos 29.018 segundos
Sonata In Gm op.49 no1, Beethoven	1 minuto 16.908 segundos
Tannhauser March, Wagner	0 minutos 12.356 segundos
Valse In Cm Op.64 no2, Chopin	1 minuto 31.763 segundos
Waltz 1 In B Op.39, Brahms	0 minutos 33.226 segundos

Tabla 3: tiempo de ejecución para k=10.

5.3. Detección de tempo.

Suponemos que la duración armónica (d_a) que más se repite coincide con el pulso de la canción, lo que nos permite deducir de manera sencilla la fórmula t=60/d_a. El problema surge cuando no existe ninguna duración armónica que se repita más que otras, sino que dos o más coinciden en la frecuencia más repetida. Para estos casos se debe tener en cuenta ciertas formulas de desempate.

La primera de estas formulas de desempate consiste en buscar, de entre todas duraciones que se repiten el mismo número de veces que la que más, grupos que produzcan tempos equivalentes, múltiplos entre ellos. Por ejemplo en el caso de la canción Bourree In Dm Fireworks Music de Handel, en algunos segmentos las duraciones armónicas que más se repiten son : 0.375, 0.5 y 0.75. Si calculamos el tempo con cada una de ellas obtenemos 160, 120 y 80. De entre estos tempos 160 es múltiplo de 80, por lo que

el tempo tiene más posibilidades de ser 160 o 80 que 120. Dentro del grupo de tempos equivalentes se seleccionará el de mayor valor.

La última fórmula de desempate que se plantea es muy sencilla y se hará uso de ella cuando la anterior no permitan calcular ningún tempo. En ese caso, de entre todas las duraciones armónicas que más se repiten, se quedaremos con la menor de ellas. Se ha comprobado experimentalmente que en un porcentaje muy alto (sobre todo en música clásica), cuando existen varias duraciones con la misma frecuencia y ninguna de ellas es múltiplo de alguna de las otras, la de duración armónica menor es la que mejor calcula el tempo.

A la hora de calcular el tempo de la canción usaremos dos métodos. En ambos realizaremos un histograma con todas las duraciones armónicas y no quedarnos con la que más se repite, usando las fórmulas de desempate anteriormente mencionadas. En el primer método, a la hora de realizar el histograma se tendrá en cuenta todos los cambios armónicos que se producen en la canción, lo que dará el tempo general. En el segundo se hará uso de los cambios armónicos de cada segmento y calculando el tempo individual de cada tramo, el tempo general es el tempo que más se repita en todos los segmentos.

Para cada método se contabilizara el porcentaje de acierto directo y el porcentaje de acierto indirecto. El primero se refiere al calculo exacto del tempo con un error permitido $d_e \pm e_m$[19] mientras que el segundo contabiliza cuando se calcula un múltiplo o submúltiplo entero del tempo original[20]. El error e_m es variable y depende de un error constante en el tempo de e_1 segundos. Por ejemplo si e_1=0.1, para un tempo de 60 el error e_m permitido será de $e_m = \pm 1$ bpm , mientras que para 120 será de $e_m = \pm 2$ bpm. Este tipo de error da un mayor margen relativo a tempos rápidos que a tempos cortos aunque en realidad el tanto por ciento de error permitido es constante ya que un margen de 1 a tempo 60 supone un 1,666 % de error el mismo que 2 para 120. Para calcular el error e_m se aplicará la siguiente fórmula: $e_m = (e_1 \cdot t) /60$

[19] error en puntos de metrónomo (aparato que marca el tempo)

[20] Solo se contabilizará los múltiplos o submúltiplos de primer orden. es decir, cuando se calcule la mita o el doble y cuatro veces o la cuarta parte del tempo original.

Por otro lado, para comprobar si el tempo calculado es múltiplo o submúltiplo del original, dividiremos el mayor de los dos entre el restante quedándonos con la parte decimal del cociente. Si ésta es mayor o igual que 1 - e_2 o menor o igual que 0 + e_2 el tempo calculado será múltiplo o submúltiplo del original.

El cálculo de un tempo múltiplo o submúltiplo del original de primer orden supone una variación en la notación musical pero no en la duración de las notas[21]. Por ejemplo (ver figura 4[22]) si para una canción de 4/4 con negra igual a 120, se ha calculado un tempo de 60 y se conoce que la subdivisión del compás es binaria, el valor de 60 puede corresponder con una negra o una blanca[23] (unidades comunes de la subdivisión binaria). Si se elige la blanca como unidad de medida la notación será idéntica, a excepción del compás que se tratará de un 2/2. Si se toma como unidad la negra toda la figuración cambia, las blancas se convierten en negras, la negras se convierten en corcheas, las corcheas en semicorcheas y así sucesivamente, de igual manera el compás se transforma de un 4/4 a un 2/4.

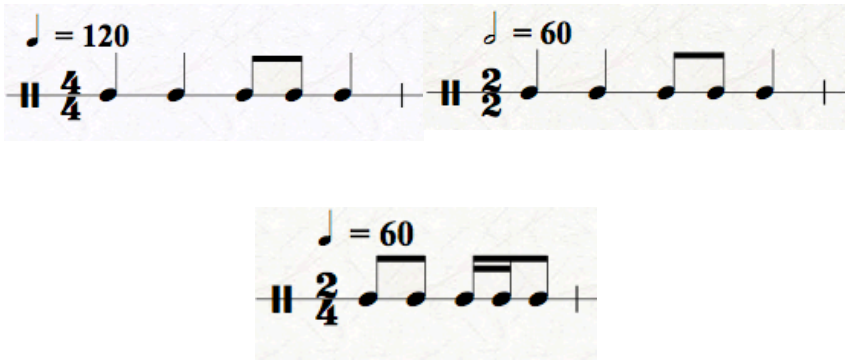


Figura 4.

[21] Esta afirmación no es cierta del todo ya que depende del funcionamiento de cada sistema de detección métrica a la hora de distinguir entre compases binarios y ternarios.

[22] En la figura solo se presenta información rítmica no armónica.

[23] Las corcheas definen el tempo cuando este es muy pequeño o en compases más complejos mientras que las redondas casi nunca se usan como unidad de medida.

Todas estas representaciones se interpretan de igual manera. Por tanto se puede concluir que detectar un tempo el doble o la mitad del tempo original, a pesar de no estar calculándose el tempo exacto, permite llegar a soluciones equivalentes. En el caso del cuádruple o un la cuarta parte, los tempos que se calculan son o muy lentos o muy rápidos, por lo que se pueden detectar de forma sencilla y transformarlos en el doble o la mitad. Esto no es válido para los casos en donde lo que se ha calculado es el triple o la tercera parte del tempo original, por lo que serán tenidos en cuenta como errores a la hora de calcular el porcentaje de acierto indirecto. Por contra estos tempos pueden ayudar a la detección de los compases de acentuación ternaria.

Una vez calculados los porcentajes de acierto directo e indirecto, se transformarán en dos porcentajes totales de acierto: El porcentaje suma de los dos porcentajes y el porcentaje de acierto total. El primero nos sirve para ver la importancia que tiene la armonía en el tempo. Aunque no es un indicador del acierto del programa a la hora de calcular el tempo si demuestra que la armonía es un factor muy importante en el tempo. El segundo mide el grado de acierto del programa a la hora de detectar el tempo. En este caso la suma entre el porcentajes no es directa si no que el indirecto se pondera en función del múltiplo del tempo que se haya calculado, de manera que los tempos que se han calculado exactamente pesarán 1, los que son la mitad o el doble 1/2 y los de un cuarto o cuatro veces 1/4. El porcentaje de acierto total se calcula sumando todas las puntuaciones de cada canción por cada género y dividiéndolas entre el número de canciones totales que lo integran.

5.3.1. Detección de tempo en MIDI.

En la tabla siguiente se muestran los mejores resultados que se han obtenido para cada género usando distintos valores de cuantificación. Las condiciones en las que se han realizado los experimentos para el cálculo del tempo en MIDI, usando los acordes base de perfecto mayor, perfecto menor, séptima de dominante, aumentados, disminuidos, 4 suspendida y 2 suspendida, son error $e_1 = 0.012$ segundos y un error e_2 del 1% ($e_2 = 0.01$). Se especifica el porcentaje de acierto directo (D), indirecto(I), sus suma (D+I), total (T) y el valor de cuantifiación usado (C) de los dos métodos:

			Tempo general				Tempo más repetido			
	Estilos	C	D	I	D + I	T	D	I	D + I	T
Clásica	Barroco	0,03	47,43%	30,35%	77,78%	62,67%	57,36%	24,79%	82,15%	69,2%
	Clasicismo	0,03	52%	36%	88%	69%	46%	42%	88%	66,5%
	Romanticismo	0,03	37,99%	38,76%	76,75%	55,26%	36,1%	41,42%	77,52%	56,62%
Jazz	Bop	0,03	10,64%	62,77%	73,4%	40,56%	13,83%	39,36%	53,19%	31,48%
	Bossanova	0	7,57%	18,18%	25,75%	15,59%	15,15%	27,27%	42,42%	27,79%
	Pre	0,03	8,43%	83,7%	92,13%	48,87%	12,92%	58,99%	71,91%	40,88%
Popular	Blues	0	9,52%	14,29%	23,81%	16,64%	16,67%	27,38%	44,05%	29,2%
	Celta	0,05	20,20%	26,26%	46,46%	30,15%	23,22%	49,45%	72,67%	47,7%
	Pop	0	18%	40%	58%	36,67%	25%	45%	71%	46,64%

Tabla 5: resultado para MIDI.

Desde el punto de vista del acierto total muestran que, a excepción de la música clásica en donde el porcentaje de acierto total medio es del 65%, no se consigue calcula el tempo de manera directa. Esto es debido a que, a excepción de la música clásica, en el resto de estilo la armonía no cambia cada pulso si no más bien cada uno o más compases, como demuestra el elevado porcentaje de acierto indirecto (en estos estilos el porcentaje de acierto indirecto es mucho mayor que el directo).

Sin embargo, estos resultados sí demuestran que en la mayoría de los estilos la armonía tiene un tempo propio que esta intrínsecamente ligado al tempo de la canción y que por lo tanto el conocimiento de este tempo armónico, así como su análisis, facilitan y ayudan a la detección del tempo rítmico. A continuación se explica detalladamente los resultados obtenidos fijándonos en la suma del porcentaje directo e indirecto que indica el tempo armónico de cada estilo.

En música clásica obtenemos que en torno a un 82,5% el tempo armónico (82,15% en barroco, 88% en clasicismo, 77,52% en romanticismo) es proporcional al tempo rítmico (y en torno al 65% se detecta dicho tempo directamente). Esto es debido a que en música clásica (en los periodos con los que se ha trabajado) se producen abundantes cambios armónicos cada pulso, salvo en piezas rápidas donde la armonía cambia cada compás y excepcionalmente en algunas canciones lentas que lo hace cada mitad del pulso. A esto hay que añadirle que la base armónica usada está especialmente diseñada para analizar este tipo de música.

En jazz los resultados no son del todo buenos puesto que la base armónica usada poco tiene que ver con la armonía de jazz. La siguiente base armónica está (ver tabla 6) especialmente diseñada para el análisis de jazz y esta compuesta por los acordes mayor con séptima(maj7) menor con séptima (m7), séptima de dominante (7), menor con séptima y la quinta disminuida (m7(b5)), disminuido con séptima (dis7) y suspendido con séptima (sus7) . La tabla siguiente muestra los intervalos que forman cada acorde referidos a C:

maj7	*m7*	*7*	*m7(b5)*	*dis 7*	*sus7*
{0,4,7, 11}	*{0,3,7, 10}*	*{0,4,7,10}*	*{0,3,6,10}*	*{0,3,6,9}*	*{0,5,7,10}*

Tabla 6.

Los resultados con la nueva base armónica usando un de e_1 = 0.01 segundos y e_2 del 1% (0.01) son :

			Tempo general				Tempo más repetido			
	Estilos	C	D	I	I + D	T	D	I	I + D	T
Jazz	Bop	0,03	12,77%	64,89%	77,66%	44,20%	17,02%	41,49	58,51%	36,55%
	Bossanova	0	10,60%	12,12%	22,72%	15,35%	15,55%	37,48%	53,03%	31,86%
	Pre	0,03	9,55%	87,64%	97,19%	52,15%	8,99%	74,16%	83,05%	45,93%

Tabla 7: resultados para MIDI usando acordes jazz

Como se puede comprobar los resultados son bastante mejores que los obtenidos con la primera base armónica, no tanto desde el punto de vista del cálculo del tempo de forma directa en donde se pasa de un 40% a un 42,7%, como el aumento de coincidencia del tempo armónico con el tempo rítmico que pasa de un 69,32% a un 75,96% y en algunos caso como en pre está cerca del 100%. Aún así, en bossanova son bastante peores que en el resto. Esto es comprensible, puesto que este estilo se caracteriza tener un ritmo constante sincopado que hace que los cambios armónicos no se produzcan en instantes proporcionales enteros al pulso, haciendo casi imposible calcular el tempo si sólo nos fijamos en los cambios armónicos. Si no se tiene en cuenta la bossanova, el porcentaje de acierto del cálculo del tempo es de 48,175% mientras que para la concordancia entre el tempo armónico y el tempo rítmico es de un 87%.

Un punto que puede llamar la atención es el bajo porcentaje de acierto en blues (ver tabla 5), ya que supuestamente su base armónica se compone de acordes de séptima de dominante. Tampoco mejora mucho usando los acordes de jazz anteriores, puesto que el porcentaje de acierto sólo es de un 30% y D+I del 47%. Frente a estos resultados y al no encontrar ninguna explicación lógica se realizó un análisis de los MIDIs de manera individual. En este análisis se comprobó que efectivamente la base armónica está formada en su mayoría por acordes de séptima de dominante pero que la duración de estos acordes es muy corta frente al bajo y sobre todo la melodía principal (normalmente un solo). Este hecho hace los resultados comprensibles puesto que lo que más va a influir en el análisis armónico[24] es la melodía. Ya de por sí es difícil realizar un análisis armónico con melodía que usan escalas tradicionales (la melodía evoluciona horizontalmente mientras que la armonía es entendida verticalmente, solo en melodía básicas es sencillo determinar la

[24] Las notas más largas tienen más influencia puesto que están presentes en mas segmentos.

tonalidad en la que se encuentran). A esto hay que añadirle que la escala básica de blues [25] es : {C, Eb, F, F#, G, Bb}, lo que dificulta el proceso mucho más. Este error es casi imposible de solucionar cambiando la base armónica, para optimizarlo se debería poder realizar un análisis conjunto entre la melodía principal en función de la escala de blues y la evolución de la armonía de la sección rítmica (sin melodía principal).

Por último, los resultados en pop no son malos del todo, 71% del tempo armónico calculado es proporcional al tempo rítmico y un 46,46% es el tempo exacto, pero son bastante peores de lo que en primera instancia se podría suponer puesto que en pop la base armónica es muy sencilla, apenas usa acordes mayores, menores y alguna séptima de dominante. En realidad, esta sencillez hace que existan pocos cambios armónicos y que las unidades armónicas sean muy grandes lo que dificulta el análisis.

En la música celta, la armonía es más complicada que en pop, pero se producen cambios armónicos con mayor frecuencia, lo que nivela ambos resultados. El problema que plantea la música celta es la cantidad de eventos que se producen en pocos segundos ya que es un estilo rápido. Esto hace que se produzcan errores si los MIDIs no están bien cuantizados. En este estilo es recomendable aumentar el margen de cuantización de los MIDIs para obtener un mejor resultados.

[25] Referida a C.

5.3.2. Detección de tempo en Audio real.

En la tabla siguiente se muestran los mejores resultados que se han obtenido para cada género usando distintos valores de cuantificación. Las condiciones en las que se han realizado los experimentos para el cálculo del tempo en las transcripciones, usando los acordes base de perfecto mayor, perfecto menor, séptima de dominante, aumentados, disminuidos, 4 suspendida y 2 suspendida, son error e_1 = 0.05 segundos y un error e_2 del 1% (e_2 = 0.07). Se especifica el porcentaje de acierto directo (D), indirecto(I), sus suma (D +I), total (T) y el valor de cuantifiación usado (C) de los dos métodos:

			Tempo general				Tempo más repetido			
	Estilos	C	D	I	D + I	T	D	I	D + I	T
Clásica	Barroco	0,05	1,79%	53,57%	55,36%	25%	1,79%	64,29%	66,07%	32,34%
	Clasicismo	0,05	2%	52%	54%	26,47%	2%	58%	60%	30,17%
	Romanticismo	0,05	1,55%	67,44%	68,99%	33,49%	1,55%	67,44%	68,99%	33,49%
Jazz	Bop	0,05	4,25%	59,57%	63,83%	31,79%	4,25%	56,38%	60,63%	30,46%
	Bossanova	0,05	7,35%	15,18%	22,53%	14,89%	12,15%	28,37%	40,52%	25,19%
	Pre	0,05	4,55%	53,03%	57,57%	28,95%	4,54%	59,09%	63,64%	32,99%
Popular	Blues	0,05	4,76%	38,1%	42,86%	21,86%	5,95%	30,95%	36,9%	20,15%
	Celta	0,05	1,01%	15,15%	16,16%	7,95%	1,01%	31,31%	32,32%	15,76%
	Pop	0,05	1%	47%	48%	23,19%	2%	51%	53%	27,13%

T*abla 8: resultado para transcripciones.*

Al igual que antes se ha repetido los experimentos para jazz usando la base armónica de jazz: maj7, m7, 7, m7(b5), dis7 y sus7. Los resultados son los siguientes:

			Tempo general				Tempo más repetido			
	Estilos	C	D	I	I + D	T	D	I	I + D	T
Jazz	Bop	0,05	5,32%	63,83%	69,15%	36,65%	2,13%	56,38%	58,51%	29,3%
	Bossanova	0,05	4,54%	53,03%	57,57%	30,05%	5,54%	59,09%	63,63%	34,09%
	Pre	0,05	2,81%	61,8%	64,61%	32,58%	3,37%	57,87%	61,24%	31,92%

Tabla 9: resultados para transcripciones usando acordes jazz

A la hora de interpretar estos resultados hay que tener en cuenta el funcionamiento de sistema de transcripción. Debido a la resolución del análisis espectral, se introduce un error base que hace que todos los resultados sean menores y que obliga a aumentar los márgenes de error. A este error base hay que añadir otros errores en función de los instrumentos que intervengan cada canción. Las transcripciones tiene bastantes errores con voz, aunque también existen problemas con instrumentos como el violín. Si añadimos estos condicionantes a los planteado anteriormente (en MIDI) se puede entender los resultados.

A continuación, se analizan los resultados en base al tempo armónico (D+I) calculado, ya que permite comprobar si este es proporcional al tempo rítmico original y se compararán los resultados con los obtenidos para las canciones en MIDI. Por último se analizarán los resultados de cálculo directo del tempo.

El valor que más diferencia presenta respecto al MIDI es la música celta que pasa de un 72,67% de acierto a 32,32%. En este tipo de música el instrumento por excelencia es el violín aunque también intervienen instrumentos de su misma familia. Como se ha comentado anteriormente, el sistema de transcripción tiene muchos problemas con el violín ,lo que produce un descenso de los porcentajes.

En música clásica el tanto por ciento medio desciende de un 82,5% en MIDI a un 62,23%. En la mayoría de canciones de música clásica interviene al menos una pista de violín lo que justifica este descenso. Por contra, también esta presente el piano, en donde el sistema transcribe mejor, lo que hace que el descenso no sea tan brusco como en la música celta.

En jazz los resultados no distan tanto unos de otros ya que el porcentaje medio para MIDI con la primera base armónica es del 69% y en la transcripción del 56%. Si se

usa la segunda base armónica, en MIDI el porcentaje aumentaba hasta el 76%, mientras que en las transcripciones lo hace hasta el 66%. Se puede apreciar como los resultados para bossanova son mejores incluso que los obtenidos para MIDI. Esto es debido a que el ritmo sincopado característico de la bosanova se difumina al realizar la transcripción.

El problema planteado en el análisis del blues sigue estando presente en la transcripción lo que hace que los porcentajes son prácticamente iguales, 44,05% en MIDI frente al 41,66% en la transcripción.

En pop los resultados se diferencian en casi 20 puntos, un 71% frente a un 53%. Al problema de los cambios armónicos muy lentos, que sigue presente, hay que añadir el error del sistema de transcripción a la hora de afrontar las partes vocales presentes en este tipo de música.

Analizando los porcentajes de acierto en el calculo del tempo se aprecia un gran descenso. Esto es debido a que casi nunca se calcula el tempo directo (los porcentajes de acierto directo medio de todos los géneros es de un 4% frente a un 49% en MIDI) lo que penaliza muchísimo. Este descenso del porcentaje directo es justificable teniendo en cuenta que es muy común que en la salida del sistema de transcripción la duración de una nota se fragmente en varias divisiones, lo que aumenta el número de puntos de segmentación en cada cuadro analizado y la posibilidad de detectar un cambio armónico donde realmente no existe ninguna nota.

5.4. Detección de tonalidad.

Uno de los ficheros de salida que genera el programa contiene la progresión armónica de la canción. En él se indica los instantes de tiempo donde se produce el cambio armónico y el tipo de acorde que se da. Mirando este fichero, la tonalidad se puede calcular de manera sencilla buscando la cadencia perfecta, formada por la progresión V-I, que más se repita. En vez de contabilizar el número de veces que se repite la cadencia para cada grado, se tendrá en cuenta la duración de la misma, y penalizará el hecho de que se encuentre la misma cadencia para distintos modos, es decir si se encuentra la cadencia perfecta para GM con una duración de 10 s y Gm con 1,01 segundos a la duración (puntuación) de GM le restaremos la de Gm.

En una primera aproximación se realizaron experimentos sobre una base de datos de 44 canciones MIDI de música clásica con diferentes tonalidades. En el 80% de los casos, la tonalidad que se calculaba era la correcta. Tras esta primera aproximación se calculó la tonalidad de la base de datos principal usando los ficheros con las progresiones armónicas calculados con un error *e1* = 0.1, un error *e2* = 0.01 y margen de cuantificación del 0.03, obteniendo los siguientes resultados:

Clásica			Jazz			Popular		
Barroco	Clasicismo	Romanticismo	Bop	Bossanova	Pre	Blues	Celta	Pop
94,65%	88%	84,5%	43,62%	50%	54,5%	26,2%	80,8%	52%

Tabla 10: detección de tonalidad MIDI.

En donde mejor se detecta la tonalidad es en la música clásica donde se obtiene un porcentaje de acierto medio del 89%. Sin embargo, para el resto de estilos, lo resultados son bastante peores a excepción de la música celta en donde el porcentaje de acierto es del 80,8%. La estructura básica del blues (rueda de blues de 12 compases) más común sigue la siguiente progresión:

I	I	I	I	IV	IV	I	I	V7	IV	I	V7/I

Tabla 11: rueda de blues.

En esta progresión no se da nunca la cadencia perfecta mayor. La mayoría de los blues derivan de esta progresión, por lo que es muy difícil calcular la tonalidad teniendo en cuenta solo esta cadencia. Si en vez de buscar la cadencia perfecta más repetida buscamos la cadencia plagal (IV - I) los resultados que se obtienen ascienden a un 75%.

En música celta se detecta la tonalidad correctamente en un 80,8% de las veces. Aunque en algunas canciones existan giros armónicos que puedan propiciar un cálculo incorrecto de la tonalidad, por regla general se trata de un género en donde mirando la cadencia perfecta se puede saber la tonalidad.

La música pop tiene un problema añadido debido a su sencillez. Es muy común el giro I-IV, que coincide con la cadencia perfecta del cuarto grado, es decir, que si en CM se produce mucho el giro C-F, el sistema de detección de tonalidad pude confundirse y dar como valor de la tonalidad el cuarto grado en vez de la fundamental puesto que el paso C-F es la cadencia perfecta de la tonalidad de FM, ya que C es el quinto grado de F.

En jazz las progresiones armónicas son más libres que en música clásica. En algunos casos incluso no se usa el acorde base de la tonalidad en ningún momento. El tipo de progresión varía muchísimo de una canción a otra (no como el blues donde todas derivan de una rueda básica) por lo que es muy difícil tratar de detectar la tonalidad usando un método standard para todas la canciones. Esto justifica los malos resultados que se obtienen dentro de este estilo, cerca de un 49% frente al 89% de música clásica, el 75% de blues o el 80,8% de música celta.

Los resultados de detección de tonalidad para las transcripciones son los siguientes (en blues se ha usado la cadencia plagal):

Barroco	Clasicismo	Romanticismo	Bop	Bossanova	Pre	Blues	Celta	Pop
91%	74%	76%	38,3%	43,2%	35,4%	62,8%	66,66%	52,5%

Tabla 12: detección de tonalidad transcripciones.

En este caso, los resultados son muy parecidos a los obtenidos en MIDI, la diferencia con estos se puede achacar al error introducido por el sistema de transcripción.

5.5. Clasificación de género.

Los resultados que presenta el artículo de Carlos Pérez-Sancho, David Rizo y José M. Iñesta (2008) se calcularon usando etiquetas de Band in a Box en el caso de los MIDIs y el sistema de Emilia Gómez Gutiérrez (2006) las transcripciones de acordes:

Tipo	Base armónica	Codificación	2 - gramas	3 - gramas	4 - gramas
Midi	Todos los acordes (5)	Grados	87 ± 4	85 ± 4	86 ± 3
		Acordes	86 ± 4	85 ± 5	85 ± 3
	Acordes cuatriadas	Grados	87 ± 4	85 ± 4	86 ± 4
		Acordes	87 ± 4	85 ± 4	85 ± 3
	Acordes triadas	Grados	84 ± 3	84 ± 4	84 ± 3
		Acordes	83 ± 4	84 ± 4	84 ± 4
Transcripción	Acordes cuatriadas	Grados	80 ± 6	82 ± 4	81 ± 5
		Acordes	89 ± 3	87 ± 3	86 ± 3
	Acordes triadas	Grados	67 ± 8	76 ± 5	74 ± 5
		Acordes	77 ± 4	79 ± 4	76 ± 5

Tabla 13. Resultado presentados en el artículo para tres géneros

Tipo	Base armónica	Codificación	2 - gramas	3 - gramas	4 - gramas
Midi	Todos los acordes (5)	Grados	41 ± 8	42 ± 10	42 ± 10
		Acordes	40 ± 10	40 ± 7	40 ± 7
	Acordes cuatriadas	Grados	51 ± 3	49 ± 7	49 ± 8
		Acordes	49 ± 5	48 ± 6	49 ± 5
	Acordes triadas	Grados	53 ± 7	54 ± 7	53 ± 9
		Acordes	55 ± 9	54 ± 11	53 ± 11
Transcripción	Acordes cuatriadas	Grados	58 ± 7	57 ± 10	56 ± 10
		Acordes	68 ± 8	64 ± 5	64 ± 5
	Acordes triadas	Grados	35 ± 11	38 ± 9	37 ± 13
		Acordes	48 ± 8	49 ± 7	47 ± 5

Tabla 14. Resultado presentados en el artículo para nueve géneros

A la hora de experimentar la clasificación de género se han usado dos bases de acordes. La usada hasta ahora en el resto de experimentos (perfecto mayor, perfecto menor, séptima de dominante, aumentados, disminuidos, 4 suspendida y 2 suspendida), y la formada por los acordes perfecto mayor (sin abreviatura, sólo la nota base del acorde), perfecto menor (m), séptima de dominante (7), séptima mayor (7M) y séptima menor (7m).

-	*m*	*7*	*M7*	*m7*
{0,4,7}	*{0,3,7}*	*{0,4,7,10}*	*{0,4,7,11}*	*{0,3,7,9}*

Tabla 15. Segundo base armónica.

Se han calculado los ficheros que contienen las progresiones tanto para midi como para transcripciones. Tras esto se ha realizado la clasificación de género usando el algoritmo de Carlos Pérez-Sancho, David Rizo y José M. Iñesta (2008) obteniendo los siguientes resultados[26]:

¿Cuantización?	Base armónica	2 - gramas	3 - gramas	4 - gramas
No	Original (5)	88 ± 4	88 ± 4	88 ± 3
	Original (7)	90 ± 3	90 ± 3	90 ± 3
	Transcripción (5)	83 ± 3	82 ± 4	77 ± 5
	Transcripción (7)	82 ± 3	81 ± 3	75 ± 4
Si	Original (5)	89 ± 3	89 ± 2	88 ± 2
	Original (7)	89 ± 3	89 ± 2	89 ± 2
	Transcripción (5)	85 ± 2	82 ± 4	77 ± 5
	Transcripción (7)	85 ± 4	80 ± 3	79 ± 4

Tabla 16. Resultados obtenidos para tres géneros.

[26] Entre paréntesis se indican la base de datos usada: 7 = 7 acordes, 5 = 5 acordes.

¿Cuantización?	Base armónica	2 - gramas	3 - gramas	4 - gramas
No	Original (5)	68 ± 4	67 ± 4	63 ± 5
	Original (7)	70 ± 6	69 ± 5	68 ± 5
	Transcripción (5)	60 ± 7	57 ± 6	52 ± 8
	Transcripción (7)	61 ± 7	58 ± 6	55 ± 8
Si	Original (5)	66 ± 6	64 ± 5	61 ± 5
	Original (7)	71 ± 4	69 ± 4	66 ± 5
	Transcripción (5)	61 ± 4	55 ± 7	51 ± 9
	Transcripción (7)	63 ± 5	60 ± 5	58 ± 6

Tabla 17. Resultados obtenidos para nueve géneros..

Los resultados obtenidos usando los ficheros de armonías de este proyecto mejoran mejoran los obtenidos con band in a box (MIDIs) obteniendo resultados experimentalmente significativos (resultado resaltado en negrita para tres géneros y los obtenidos para las canciones originales en 9 géneros), que indican que los porcentajes obtenidos suponen una mejora considerable con respecto a los planteados en el artículo. Con respecto a los resultados obtenidos para las transcripciones de los MIDIs, los resultados son similares a los calculados en el artículo usando el sistema de Emilia Gómez Gutiérrez (2006).

6. Conclusiones y futuras líneas de investigación.

6.1. Conclusiones.

El algoritmo implementado en este proyecto segmenta una canción en función de sus unidades armónicas. Una vez conocida la evolución armónica que sigue cada canción se ha calculado su tempo identificando la duración armónica que más se repite con el pulso de la canción. Los resultados demuestran que la armonía es punto clave ya que en un porcentaje muy elevado el tempo armónico (la unidad, el doble o la mitad, el triple o un tercio, el cuádruple o un cuarto) calculado es proporcional a la tempo rítmico. Por otra parte, los porcentajes totales de acierto en el cálculo del tempo concluyen que la armonía por sí sola no permite calcular el tempo de manera exacta puesto que, a excepción de la música clásica (con un porcentaje de acierto directo del 65%), en el resto de estilos el porcentaje de acierto directo es inferior.

El análisis armónico calculado se ha usado para la clasificación de género de la base de datos descrita en el artículo de Carlos Pérez-Sancho, David Rizo y José M. Iñesta (2008) mejorando los resultados obtenidos para los MIDI, y alcanzando resultados estadísticamente significativos para la clasificación en tres y nueve géneros, y siendo muy parecidos los de las transcripciones. También se ha usado el análisis armónico para calcular la tonalidad, obteniendo buenos resultados en música clásica, (89% de acierto), celta, (81%) y blues, (75%).

Por tanto se puede concluir que, a falta de determinar el grado exacto de acierto en el cálculo de progresiones armónicas, el algoritmo es válido para clasificación de género y detección de tonalidad. Los resultados de detección de tempo hacen pensar que un sistema híbrido entre el tempo armónico y los onsets es la clave para la correcta extracción del tempo final, sin dejar de lado las posibilidades que ofrece a la hora de calcular la métrica, detección de frases, estructuras, motivos, etc.

6.2. Futuras líneas de investigación.

Este proyecto supone una primera aproximación en el cálculo del tempo en base a la armonía que ni mucho menos queda resulto, pero sí cimienta los principios para futuras investigaciones. A continuación se plantean una serie de posibles mejoras que se pueden llevar a cabo en un futuro desde el punto de vista de análisis del tempo, tonalidad y clasificación de género.

Un trabajo esencial es la creación de una base de acordes distintos para cada uno de los géneros: barroco, clasicismo, romanticismo, bop, bossanova, pre, blues, celta y pop, que maximicen los resultados. También es interesante ampliar las bases de géneros con los que se va a trabajar de manera que se pueda afrontar rock, funky, country, electrónica, heavy, diferentes estilos de músicas étnicas, africanas, asiáticas, etc. Un apartado que queda pendiente es realizar una comprobación de los cambios armónicos detectados con la progresión original, de manera que se obtenga un porcentaje de acierto que valide el sistema como detector de progresiones armónicas. Por otro lado, es necesario establecer el margen a por debajo del cual se eliminarán unidades armónicas para optimizar los resultados. Por último, es necesario definir los motivos exactos que hacen que el funcionamiento en pop sea peor del esperado, discerniendo si el problema es de las características musicales del estilo o del funcionamiento del programa.

Desde el punto de vista de detección de tonalidad, es necesario ampliar el método de manera que se pueda afrontar con garantías todos los estilos, especialmente el jazz. Para ello se plantea realizar un estudio de los parámetros que definen la tonalidad, especialmente las cadencias, para cada uno de los estilos que se va abordar y tratar de obtener algún descriptor común para todos los géneros o como mínimo descriptores significativos para cada uno ellos.

Un punto muy importante es determinar el tempo correcto, en vez del doble/mitad, el triple/un tercio o el cuádruple/un cuarto para tener un sistema de detección de tiempo fiable. Para tratar de resolver este apartado se puede hacer uso de las progresiones armónicas ya calculadas, mirando la duración de los motivos armónicos que más se repiten en cada canción y comparándolos con el tempo calculado, o bien usar los detectores de tempo en base a la energía o por último creando un sistema híbrido entre ambos.

Posteriormente, se afrontaría el calculo de la métrica con garantías, para la que se puede hacer uso de los cambios armónicos (motivos armónicos que más se repiten), onsets o un sistema híbrido entre ambos.

6.3. Coda.

La música es la disciplina artística abstracta por excelencia. En ella todos los elementos de control, manejo e interpretación se funden y difuminan por mucho que nos empeñemos en clasificarlos y ordenarlos en conceptos como armonía, tempo, métrica, género ... quedando exclusivamente como límite la imaginación de cada músico. A día de hoy todavía no se tiene clara la influencia recíproca de cada uno de los parámetros (cada vez más en aumento) que definen una obra musical. Por esto, ningún parámetro debe ser dejado de lado en la búsqueda del resto. Este proyecto viene a demostrar esta idea, puesto que a priori la armonía tiene poco que decir en parámetros rítmicos como el tempo y sin embargo, se ha demostrado que la mayoría de los tempos armónicos calculados son proporcionales al tempo rítmico.

7. Bibliografía.

Antonio Pertusa y José M. Iñesta “Multiple fundamental frequency estimation using Gaussian Smoothness” *IEEE International Conference on Acoustics, Speech and Signal Processing: ICASSP* 2008.

Benoit Meudic “Automatic meter extraction from MIDI files” *CiteSeerX - Scientific Literature Digital Library and Search Engine,* 2002.

Bryan Pardo y William P. Birminghan “Algorithms for Chordal Analysis” *Computer Music Journal,* 26:2, pp. 27-49, 2002.

Carlos Pérez-Sancho, David Rizo y José M. Iñesta. “Genre classification using chords and stochastic language models”. Pattern Recognition and Artificial Intelligence Group Departamento de Lenguajes y Sistemas Informáticos Universidad de Alicante, Ap. 99, E-03080 Alicante, Spain, 2008.

David Temperley and Daniel Sleator “Modeling meter and harmony: A preference-rule approach” *Computer Music Journal,* 23:1, pp. 10-27, 1999.

David Temperley and Daniel Sleator “The Melisma Music Analyzer” *Cognition of Basic Musical Structures,* 2004.

Emilia Gómez Gutiérrez “Tonal Description of music audio signals” *Tesis Doctoral, 2006*

Eric D. Scheirer “Tempo and beat analysis of acoustic musical signals” *J. Acoust. Soc. Am. Volume 103, Issue* 1, pp. 588-601, 1998.

Fabien Gouyon y Simon Dixon “A review of rhythm description systems” *Computer Music Journal,* 29:1, 2005.

Fabien Gouyon “Pulse-dependent analyses of percussive music” *AES 22nd.* 2002.

Frank Seifert, Katharina Rasch y Michael Rentzsch “Tempo Induction by Stream-Based Evaluation of Musical Events” *University of Victoria, Germany,* 2006

Geoffroy Peeters “Time variable Tempo detection and beat marking” *ICMC, Barcelona, España,* 2005.

Geoffroy Peeters “Tempo detection and beat marking for perceptual tempo induction” *ICMC, Barcelona, España,* 2005.

Heinrich Taube “Automatic Tonal Analysis: Toward the Implementation of a Music Theory Workbench” *Computer Music Journal*, 23:4, pp. 18-32, 1999.

Jean-François Paiement, Douglas Eck y Samy Bengio “A probabilistic Model for Chord Progresssions” *Queen Mary, University of London* 2005.

Klaus Frieler “Beat and meter Extraction using Gaussified onsets” *Universitat Pompeu Fabra*, 2004.

Matthias Varewyck y Jean-Pierre Martens “Assessment of state-of-the-art meter analysis systems with an extended meter description model” *Austrian Computer Society (OCG),* 2007.

Ruohua Zou y Joshua D.Reiss “ Music onset detection combining energy-based and pitch-based approaches” *Queen Mary, University of London* 2007.

Simon Dixon “Automatic extraction of tempo and beat from expressive performances” *JNMR, 30(1):* 39-58, 2001.

Walter Piston “Armonia” *SpanPress Universitaria,* 1998.

Diether de la Motte “Armonía”, *Labor,* 1989.

8. Agradecimientos.

Esta investigación no se podría haber llevado a cabo sin la colaboración y paciencia de mucha gente. En primer nombrar a Antonio Pertusa, por contestar a un número infinito de correos y estar siempre dispuesto a solucionar cualquier problema. Al departamento de Lenguajes y Sistemas Informático, por su disponibilidad en todo momento. A Carlos Pérez-Sancho, por su ayuda con el sistema de transcripción.

A mi toda familia por su apoyo, compresión, ánimos, paciencia, y mil más, especialmente a mi padre transmitirme la pasión por la música. A mi madre y hermanos por todo el cariño (y el follón) que me han dado, siempre presente en el momento oportuno. A mi primo por los partidos de fútbol del miércoles. A mis abuelas, por los gritos en la escalera y el arroz con leche y a mis abuelos, por haberlos disfrutado tanto.

A Álvaro por sus comisiones (¿y que haremos mañana?), a Miguel Angel por acceder a tocar con un manco y compartir su talento con migo y al resto de mis amigos, porque en esta vida siempre hay tiempo para una cerveza.

Y muy especialmente a Elena por ser sueño y realidad.

9. Apéndices.

9.1. Segmentación armónica y detección de tempo.

```
/*PROYECTO FINAL DE CARRERA:

DETECCIÓN DE ARMONÍA Y ESTRUCTURA RÍTMICA MUSICAL

        GABRIEL MESEGUER BROCAL*/

//LIBRERIAS
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include <string.h>
#include "graph.h"

using namespace std;

//CONSTANTES
const string chord[]=
{"C","C#","D","Eb","E","F","F#","G","G#","A","Bb","B"}; //Acordes
const float error=0.003; // Margen de error para el histograma
const int kfrg=10;


/*                  -------------AYUDA------------
--Variables vector: se definen como vector <tipo del vector> nombre; es
un vector de tamaño varible del tipo que nosotros decidamos, si no
definimos el tamaño solo tiene un elemento.
--push_back: variable.push_back(valor) añade "valor" a la ultima casilla
de "variable"
--resize:  variable.resize(tam) da a "variable" el tamaño "tam"
--size(): devuelve el tamaño del vector
--clear(): limpia la información almacenada en un vector*/

/*DEFINICIÓN DE ESTRUCTURAS Y OTROS*/

struct notaux{   /*Se utiliza para almacenar la información del fichero
de entrada*/
    int pitch;  //Notas
    float tini; //Inicio de las notas
    float dur;  //Duración
};
```

```
struct nota{/*Se almacena la información del fichero de texto una vez
ordenada en tiempo, es utilizada en pasos intermedios */
    int pitch; //Notas
    float t;   //Tiempo en el que se produce un evento
    int e;     //Tipo de evento: 0 final de nota 1 inicio
};

struct pianola{ /*Se alamcena la información para trabajar con ella el
resto del programa. Tiene forma de pianola con la salvedad de que las
notas no están en notación clásica si no en pitch, es decir de 0 a 128*/
    float t;          //Tiempo en donde se produce un evento
    int notas[128];   //Notas presentes en el evento
};

bool operator< (const nota & left, const nota & right){/* Ordena
vectores. Es usada por la función sort, indica que un vector se debe
ordenar de menor a mayor tiempo y en caso de existir instante de tiempos
iguales, en función de su estado el 0 (final de nota) prima sobre el 1
(inicio de nota).*/
    bool salida=false;
    if (left.t<right.t) salida=true;
    if (left.t==right.t){
        if (left.e<right.e) salida=true;
    }
    return salida;
}

/*-------------------FUNCIONES-----------------------
EL PROGRAMA SE DIVIDE EN DOS PARTES:
1) Lectura del fichero de entrada y ordenación de la información.
2) Segmentación, detección de tempo.*/


/*SEGUNDA PARTE. SEGMENTACIÓN, DETECCIÓN TEMPO:*/


/*FUNCIÓN NOMBRE: Se encargada de formalizar el nombre del acorde y el
tipo.Se basa en las constantes definidas al inicio del programa y los
parámetros que recibe de la función pesos que le indican de que acorde se
trata y el tipo.*/
string
nombre(vector <string> mode, string notacion, int a, int e){
    notacion=chord[a]+mode[e];
    return(notacion);
}
```

```
/*FUNCIÓN ESCRITURAFICHERO: escribe los resultados calculados*/
void
escriturafichero (vector <float> tc, float tg, float t, float tb, vector
<string> progresion, ofstream &f, ofstream &ft){
    if (f.is_open()){
        for(int i=0; i<(int)tc.size()-1; i++){
                f<<tc[i]<<" "<<progresion[i] <<endl;
        }
        f<<tc[tc.size()-1];
        ft << "Tempo general = "<< tg<<endl;
        ft << "Tempo más repetido en los segmentos = "<< t<<endl;
        ft << "Tempo general con beat = "<< tb<<endl;
    }else{
        cout << "Fichero no encontrado";
    }
}

/*FUNCIÓN UNO: Criterio de despempate.Buscar entre todos los tempos que
se pueden calcular con las duraciones que si existen alguno múltiplo de
otro*/
void
uno(vector <float> repe, float &tempo, int a){
    int cont=0;
    vector <float> tempos;
    for (int i=0; i<(int)repe.size(); i++){/*Calculo de todos los tempos
    sobre los que se tiene que desempatar*/
        if (a==1) tempos.push_back((((float)((int)round(6000/repe[i])))
                                                        /(float)100));
        else tempos.push_back(repe[i]);
    }
    sort(tempos.begin(), tempos.end()); /*ordenamos los tempos de menor a
                                        mayor*/
    for(int i=(int)tempos.size()-1; i>=0; i--){
        for(int j=i-1; j>=0; j--){
            float t=((float)((int)round(100*(tempos[i]/tempos[j]))))
                                                        /float(100), r;
            int s=t;
            r=t-s;
            if(r==0){/*Un tempo que es exactamente múltiplo de otro prima
                            sobre los calculados con un error permitido*/
                tempo=tempos[i]; cont=1;
            }else{//Tempos con error permitido
                if(r>=0 && r<=0.125 && cont==0) tempo=tempos[i];
            }
        }
    }
}
```

```
/*FUNCIÓN HC: Es la función que se encarga de realizar las operaciones
para el calculo el histograma*/
vector< pair<float,int> >
hc(vector <float> a, float e){
    float aux;
    vector< pair<float,int> > h;
    while((int)a.size()!=0){
        int r=0, i=0; aux=a[0];
        while (i<(int)a.size()){/*Buscamos repeticiones que se repiten y
                                  las eliminamos del vector dur*/
            if (a[i]>aux-e && a[i]<aux+e){
                r++;
                a.erase(a.begin()+i);/*Se eliminan la duración que se
                repite*/
            }else i++;/*Contardor que permite recorrer el vector
                dur*/
        }
        h.push_back(make_pair(aux,r)); /*Se almacena la duración
                analizada y el número de veces que se repite*/
    }
    sort(h.begin(),h.end());
    return(h);
}

/*FUNCIÓN TEMPO2 calcula un segundo tempo general como el tempo que más
se repite entre el tempo de cada segmento*/
float
tempo2(vector<float> tiempos){
    vector<float> repe;
    float t=0, cont=0;
    vector< pair<float,int> > h;
    h=hc(tiempos, 1);
    for (int o=0; o<(int)h.size(); o++){
        if(h[o].second>cont)cont=h[o].second; /*Duración que más se
                                               repite*/
    }
    for (int o=0; o<(int)h.size(); o++){
         if(h[o].second==cont) repe.push_back(h[o].first);  //Comprobamos
                                    que no se repita mas de una vez
    }
    if((int)repe.size()==1) t=repe[0]; /*Si solo se repite una vez
                                calculamos el tempo directamente*/
    else{ //Formulas de desempate
        uno(repe, t, 0);
        if(t==0) t=repe[0];
    }
    return(t);
}
```

```
/*FUNCIÓN HISTOCAL: Calcula el histograma de las duraciones armónicas*/
float
histocal(vector<float> tt){
    int p=0;
    float tempo=0, d=tt[tt.size()-1]-tt[0]; /*"d" duración total del
                                                segmento*/
    vector <float> dur, repe;
    vector< pair<float,int> > h;
    for (int i=1; i<(int)tt.size(); i++){ /*Calculo de todas las
                                                duraciones*/
        float daux=tt[i]-tt[i-1];
        if (daux<0) daux=daux*(-1);
        if(daux>0.00009)dur.push_back(daux); /*eliminamos la duraciones
                                        de armónicas muy pequeñas*/
    }
    h=hc(dur, error);
    for (int i=0; i<(int)h.size(); i++){
        if(h[i].second>p && h[i].first!=0)  p=h[i].second; /*Se busca la
                                duración que más se repite*/
    }
    for (int i=0; i<(int)h.size(); i++){
         if(p==h[i].second)  repe.push_back(h[i].first); //Comprobamos
                                        que no se repita mas de una vez
    }
    if((int)repe.size()==1){
        tempo=(((float)((int)round(6000/repe[0])))/(float)100); //Si solo
        se repite una vez calculamos el tempo directamente
    }else{
        uno(repe, tempo, 1); //Formulas de desempate
        if(tempo==0) tempo=(((float)((int)round(6000/repe[0])))
                                                        /(float)100);
    }
    return(tempo);
}
```

```
/*FUNCION HISTOGRAMA: Calcula el tempo y comprueba si los cambios
armónicos detectados se dan en onset o offset*/
float
histograma(vector<float> &th, vector <float> &tonset, int a){/*Recibe los
instantes de tiempo en donde se produce un cambio armónico almacenados en
th del segmento analizado*/
    float aux=0, tempo=0;
    vector <float> tg1(th.begin(), th.end()), tg2, tee;
    copy (th.begin(), th.end(), tg1.begin());
    if(a==0){
        for (int i=0; i<(int)tg1.size(); i++){ /*Nos quedamos solo con
                    los instantes de tiempo donde se produce un onset*/
            aux=0;
            for (int j=0; j<(int)tonset.size(); j++){
                if(tg1[i]<tonset[j]+error && tg1[i]>tonset[j]-error){
                        aux=1;
                }
            }
            if (aux==1) tg2.push_back(tg1[i]);
        }
    }
    if((int)tg2.size()>3 && a==0) tempo=histocal(tg2);  /*Cuando se más
                        de tres cambios armónicos en inicio de onset*/
    else tempo=histocal(tg1); /* En caso contrario se usa todo el
                                        segmento*/
    while(tempo<40 && tempo!=0) tempo=tempo*2;
    while(tempo>300 && tempo!=0) tempo=tempo/2;
    tempo=(((float)((int)round(tempo*100)))/(float)100);
    return(tempo);
}

/*FUNCION BEAT: Calcula los onsets donde se produce beats en función del
tempo general que se calcula en función de la armonía*/
float
beat (float t, vector<float> onset, ofstream &fb){
    float frame=60/t, tempo;
    vector<float> on;
    for(float i=onset[0]; i<onset[(int)(onset.size()-1)]; i=i+frame){
        for (int j=0; j<(int)onset.size(); j++){
            if(onset[j]<i+0.05 && onset[j]>i-0.05){
                on.push_back(onset[j]);
                fb<<onset[j]<<endl;
            }
        }
    }
    tempo=histograma(on, onset, 1);
    return(tempo);
}
```

```
/*FUNCIÓN CÁLCULOS: Se encarga de realizar todos los cálculos necesarios
para obtener el peso, usando la fórmula Peso = P - (M+N). P representa el
número de veces que las notas del acorde están presentes en el segmento
analizado. M representa las notas del acorde que no aparecen en el
segmento analizado. N representa el número de notas ajenas al acorde que
aparecen en el segmento analizado.*/
int
calculos (int n[], int r, int a, vector <vector <int> > &Acorde){/*En "n
[]" tenemos las notas presentes en el segmento analizado. "r" es el
"factor de transporte" que indica la altura del acorde de 0 a 11. "a"
indica el tipo de acorde de que se trata. En Acorde tenemos todos los
tipos de acordes referidos a C*/
    int peso=0, P=0, N=0, M=0, aux[12];
    vector <int> chord(Acorde[a].size());
    for (int i=0; i<12; i++) aux[i]=n[i]; /*Copiamos el segmento a
                        analizar, para no modificar el original*/
    for (int i=0; i<(int)Acorde[a].size(); i++){
        chord[i]=Acorde[a][i]+r; /* En cada iteración calculamos la nota
                        del acorde con el acorde que probamos sumando r*/
        while (chord[i]>11) chord[i]=chord[i]-12; /*Si se sale de rango
                                                (0 a 11) restamos 12*/
        if (n[chord[i]]>0){ /* Si la nota del acorde a tratar esta
                presente en el intervalo de tiempo calculamos P y la
                eliminamos de la copia de las notas que están sonando*/
            P=P+n[chord[i]];
            aux[chord[i]]=0;
        }else{  /*En caso de que la nota no se encuentre en el intervalo
                de tiempo incrementamos M*/
            M++;
        }
    }
    for (int j=0; j<12; j++) N=N+aux[j]; /*Como se ha ido eliminando las
                    notas que estaban presentes en el acorde el cálculo
                    de N se reduce a la suma del resto de notas*/
    peso= P - ( M + N );  // Aplicamos la formula del peso
    return(peso);
}
```

```
/*FUNCIÓN PESOS: Recorre el vector de notas presentes en cada intervalo a
analizarque recibe de la función segmentación en el n[] y devuelve el
pesos y el nombre delacorde correspondiente*/
```

```
float
pesos(int n[], string &notacion, vector <vector <int> > &Acordes, vector
<string> &mode){
    int aux=0, no, d;
    float peso=0, paux=0, weights[Acordes.size()];
    for (int i=0; i<12; i++){ /*El índice "i" indica la altura del acorde
                                        {C, C#, D, (...), B}*/
        if (n[i]>0){ /*Siempre que una nota este presente en el segmento
                    a analizar se probara con todos sus acordes bases*/
            for (int j=0; j<Acordes.size(); j++){ /*El índice "k" indica
                                                    el tipo de acorde*/
                weights[j]=calculos(n,i,j, Acordes);
                if(j==0) paux=weights[j]; /*Almacenamos la primera
                            iteración para compararla con las demás*/
                if(weights[j]>paux){
                    paux=weights[j]; aux=j;
                }
            }
        }
        if (i==0){/*Almacenamos la primera iteración para compararla con
                    las demás*/
            peso=paux; d=aux; no=i;
        }
        if (paux>peso ||(paux==peso && aux<d)){
            peso=paux; d=aux; no=i;
        }
    }
    notacion=nombre(mode, notacion, no ,d);
    return(peso);
}


/*FUNCIÓN INTERCAMBIO: Traduce la información del caminomaslargo en
instantes de tiempo y su correspondiente armonía*/
void
infocamino(vector<int> &caminomaslargo, vector<float> &th, vector<string>
&ac, vector<Edge> &edges, vector<string> &prog, vector<pianola> &pfr,
string ant){
/*En "caminomaslargo" tenemos los vértices que forman el camino más
largo. En "th" se almacenan los instantes donde existe un cambio
armónico. En "prog" tenemos la matriz correspondiente al grafo con todos
los acordes detectados para todos los segementos. "edges" contiene todos
los caminos que existen. "pfr" contiene toda la información del segmento
a analizar. "tc" y "progresion" son variables globales, en donde se
almacena todas los instantes donde se produce un cambio armónico y su
acorde correspondiente*/
```

```
    for(int i=1; i<(int)caminomaslargo.size(); i++){
        for (int j=0; j<(int)edges.size(); j++) {/*Búsqueda de los
            caminos que forman el camino mas largo. El valor del índice j
            que corresponde con el caminos más largo indica en que
            casilla se encuentra el acorde asociado en prog*/
            if(caminomaslargo[i-1]==edges[j].first &&
                                caminomaslargo[i]==edges[j].second){
                if (ac.size()==0 && prog[j]!=ant){
                    th.push_back(pfr[caminomaslargo[i-1]].t);/*La
                                traducción del camino más largo a
                                instantes de tiempo es inmediatos*/
                    ac.push_back(prog[j]);
                }
                if(ac.size()>0){ /*Comprobación del último acorde del
                                segmento anterior con el primero
                                    del que se está calculando*/
                    if(prog[j]!=ac[ac.size()-1]){
                        th.push_back(pfr[caminomaslargo[i-1]].t);/*La
                                traducción del camino más largo a
                                instantes de tiempo es inmediatos*/
                        ac.push_back(prog[j]);
                    }
                }
            }
        }
    }
    th.push_back(pfr[caminomaslargo[caminomaslargo.size()-1]].t);
}


/*FUNCIÓN SEGMENTACIÓN: Fragmenta las trozos de canción en unidades
armónicas sin necesidad de conocer el tempo, métrica, tonalidad, u otros
conceptos musicales.No realiza casi cálculos, es la encargada de llamar
al resto de funciones que los realizan, ordenar la información que
devuelven. Genera la variables necesarias para la construcción del grafo,
el calculo del tempo y la escritura en los fichero*/
float
segmentacion(vector<pianola> &pfr, char nombre [], ofstream &ft,
vector<string>  &progresion, vector <vector <int> > &Acordes, vector
<string> &mode, vector <float> &tc, vector <float> &tiempos, vector
<float> &tonset, int e, int &co){
    int aux, c, n[12];
    string notacion, ant;/*"notacion" se almacena el nombre de los
                            acordes, "ant" variable auxiliar*/
    vector<Edge> edges; //Caminos del grafo
    vector<string> nodenames, prog, ac; /*"nodenames" nombre de los
                                nodos, "prog" y "ac" acordes hallados*/
```

```
vector<float> weights, th; /*weights = Pesos, "th"  Almacena los
                    instantes donde se produce un cambio armónico*/
//---------------------
for(int z=0; z<(int)pfr.size(); z++){
    stringstream os;  //Nodo en el que se esta trabajando.
    os << z;
    nodenames.push_back(os.str());
    for(int o=0; o<12; o++) n[o]=0; //Inicialización del segmento
    for(int i=z; i<(int)pfr.size()-1; i++){
        for(int j=0; j<128; j++){/*Paso de los pitch a notas,
        perdiendo la información de octava: El resultado es un vector
        de 12 elementos de 0 a 11, cada elemento representa una nota
        C es 0 y B 11  el resto se calcula cromáticamente, el número
        de cada elemento indica el n° de veces que se repite la nota
        en el segmento seleccionado*/
            aux=pfr[i].notas[j];
            if(aux==1){
                c=j%12;
                n[c]=n[c]+1; /*n = notas que están "sonando" en un
                                determinado intervalo de tiempo*/
                }
        }
        float peso = pesos(n, notacion, Acordes, mode);/*Se almacena
            el peso en "peso" y el acorde que le corresponde en
                                                    "notacion"*/
        Edge e(z,i+1); // Calculamos el camino correspondiente
        edges.push_back(e);
        weights.push_back(peso);
        prog.push_back(notacion);
    }
}
/*longestpath calcula el camino mas largo, devolviendo los nodos
intermedios que lo componen: está en graph.h*/
vector<int> caminomaslargo=longestpath(&nodenames[0], (int)
nodenames.size(), &edges[0], &weights[0], (int)edges.size(), 1);
/*Contempla los casos en que el camino más largo va del vértice
inicial al final*/
if((int)caminomaslargo.size()==0) caminomaslargo.push_back(0);
//Se añade el primer y último vértice.
if((int)caminomaslargo[0]!=0){
     caminomaslargo.insert(caminomaslargo.begin(), 0);
}
if(progresion.size()>0) ant=progresion[progresion.size()-1]; /*ant es
        el contador del último acorde que se calculo en el segmento
                                                    anterior*/
caminomaslargo.push_back(pfr.size()-1);
/*Traducción de la información del grafo a instantes de tiempo y
nombre de acordes.*
```

```
    infocamino(caminomaslargo, th, ac, edges, prog, pfr, ant);
    /* Almacenamos los tiempos de cambios armónicos, la progresión y el
    tempo de cada segmento*/
    if((int)th.size()>2 || e==1){
        co=1;
        for(int i=1; i<(int)th.size()-1+e; i++){
            tc.push_back(th[i]);
            progresion.push_back(ac[i-1]);
        }
        if (ft.is_open()){
            float tempo=histograma(th, tonset, 0);
                ft << th[0]<< "  Tempo = "<< tempo << endl;
                tiempos.push_back(tempo);
        }else{
            cout << "Fichero no encontrado";
        }
    }else co++;
    return(th[th.size()-2+e]);
}


/*PRIMERA PARTE. LECTURA DEL FICHERO DE TEXTO Y ORDENACIÓN DE LA
INFORMACIÓN:*/

/*FUNCIÓN PIANOL: Esta función se encarga de transformar la información
almacenada en notas en forma de pianola. Se ve que notas están sonando en
un instante de tiempo determinado y las almacenamos en s. Cuando se
cambia de instante se copia el valor de s en el vector p (vector final) y
se pasa al instante de tiempo siguiente. Se sigue almacenado las notas
que están sonando en s, que contiene la información del instante
anterior. Una nota dejará de sonar cuando aparezca un estado 0*/
vector<pianola>
pianol(vector<nota> &notas, float cuant){
    vector<pianola> p(1);
    int aux, cont=0, s[128]; /*cont es el encargado de generar el nuevo
    vector de tiempo*/
    for(int i=0; i<128; i++) s[i]=0;
    for(int i=0; i<(int)notas.size(); i++){
        if(i==0){//Primera iteración
            p[cont].t=notas[i].t;
            aux=notas[i].pitch;
            s[aux]=1;
        }else{
            /*Si se sigue en el mismo instante de tiempo almacenamos las
            notas que esten sonando. cuant = valor de la cuantificación*/
            if(p[cont].t>= notas[i].t - cuant &&
                              p[cont].t <= notas[i].t + cuant){
                aux=notas[i].pitch;
```

```
                if(notas[i].e==1) s[aux]=1;
                else s[aux]=0;
            }else{
                for(int j=0; j<128; j++){/*Se copia las notas que están
                                           sonando en instante anterior
                                               en el vector p final*/
                    if(s[j]==1) p[cont].notas[j]=1;
                }
                /*Primera iteración del
                siguiente instante de tiempo*/
                p.resize(p.size()+1); cont++;
                p[cont].t=notas[i].t;
                aux=notas[i].pitch;
                if(notas[i].e==1) s[aux]=1;
                else s[aux]=0;
            }
        }
    }
    return(p);
}


/*FUNCIÓN VECTORTIEMPO: transforma la información almacenada en
vector<notaux> notes ordenandola como:
tiempo en el que se produce un evento, evento que se produce y tipo de
evento. Al hablar de evento que se produce se hace referencia a las notas
en notación pitch, mientras que al hacerlo del tipo de evento se refiere
a si se produce una nota on o una nota off. */
vector<nota>
vectortiempo(vector<notaux> &notes){
    vector<nota> aux (notes.size()), notas (notes.size());
    for(int i=0; i<(int)notes.size(); i++){
        notas[i].pitch =notes[i].pitch;
        aux[i].pitch =notes[i].pitch;
        notas[i].t = notes[i].tini;
        aux[i].t = notes[i].tini + notes[i].dur;
        notas[i].e = 1; // nota on
        aux[i].e = 0;   // nota off
    }
    for(int i=0; i<(int)aux.size();i++) notas.push_back(aux[i]);
    sort(notas.begin(), notas.end());/*sort devuelve el vector ordenado
                                           en función del tiempo.*/
    return (notas);
}
```

```
/*FUNCIÓN PROCESARLINEA: se encarga de procesar la líneas del fichero que
contien los acordes base*/
void
procesarLinea(string s, vector<string> &mode, vector< vector <int> >
&Acordes){
    istringstream iss(s);
    string token;
    int primero=true;
    vector <int > chord;
    while(getline(iss, token,' ')){
        if (primero)mode.push_back(token);
        else chord.push_back(atoi(token.c_str()));
        primero=false;
    }
    Acordes.push_back(chord);
}


/*FUNCIÓN LEERFICHERO: Esta función se encarga de leer el fichero de
entrada y almacenar la información en las variables tini, dur y pitch que
corresponden respectivamente al tiempo de inicio, duración y altura de
una nota. El fichero de texto tiene que estar ordenado de la siguiente
manera: tiempo de inicio de la nota, duración de la misma, pitch y solo
debe contener estan información*/
void
leerfichero(char v[], vector<notaux> &notes, char a[], vector<string>
&mode, vector< vector <int> > &Acordes){
    fstream f, fa;
    f.open(v, ios::in);
    fa.open(a, ios::in);
    //Lectura de la información de entrada
    if(f.is_open()){
        float tini, dur;
        int pitch;
        f>>tini>>dur>>pitch;
        while(!f.eof()){
                notaux n;
                n.tini=tini;
                n.dur=dur;
                n.pitch=pitch;
                notes.push_back(n);
                f>>tini>>dur>>pitch;
        }
        f.close();
    }else cout<< "Error: Fichero de entrada no encontrado"<<endl;
```

```
    //Lectura de los acordes base
    if (fa.is_open()){
        string s;
        getline(fa,s);
        while (!fa.eof()){
            procesarLinea(s,mode,Acordes);
            getline(fa,s);
        }
        fa.close();
    }else cout<< "Error: Fichero de acordes  no encontrado"<<endl;
}


/*ORGANIZACIÓN DE LA INFORMACIÓN*/

/*FUNCIÓN  PRINCIPAL: realizar las llamadas al resto de funciones
importantes, es donde se definen las variables básicas para cada proceso
y donde se realiza la división de la canción en subconjunto para ser
analizados de forma independiente.*/
void
principal (char fichero[], char nombre[],  char acordes[], float cuant){
    int co=1;
    float cont=0, aux=0, tg;    //"cuant" valor de la cuantificación
    //Variables utilizadas en la lectura de la información
    vector<notaux> notes;
    vector<nota> notas;
    vector<pianola> p, pfr;
    vector <vector <int> > Acordes;
    //--------------------------
    vector<string> progresion, mode; /*progresión armónica de toda la
                                                          canción*/
    vector <float> tc, tiempos, tonset, duracion; /*"tc" instantes de
                tiempo donde se detecta un cambio armónicos de toda la
                canción "tiempos" tiempos calculados en cada segmento,
                "tonset" tiempos donde se produce un onset, "duracion"
                                    duración total de la canción*/
    //---Lectura de la información---
    leerfichero(fichero, notes, acordes, mode, Acordes);
    for (int i=1; i<(int)notes.size(); i++){
        if(notes[i-1].tini!=notes[i].tini){
             tonset.push_back(notes[i-1].tini);//Onset de la canción
        }
    }
    notas = vectortiempo(notes);
    p = pianol(notas, cuant); // "p" = pianola

    //--------------------------
```

```
    /*---Ficheros:  f-> evolución armónica, ft-> tempos segmentos, fb->
    tempos beat ---*/
    ofstream f, ft, fb;
    string cad1=string(nombre)+"-Tempos.txt";
    string cad2=string(nombre) + "-Armonia.txt";
    string cad3=string(nombre) + "-beat.txt";
    ft.open(cad1.c_str(), ios::app);
    f.open(cad2.c_str(), ios::app);
     fb.open(cad3.c_str(), ios::app);
    //División de la canción en subconjuntos dependientes de Kfrg
    tc.push_back(p[0].t);
    while (aux<p[p.size()-1].t){
        if(p[cont].t+co*kfrg<p[p.size()-1].t &&
                                (p.size()-cont)>100*co){ //Caso general
            for(int i=cont; p[i].t<p[cont].t+kfrg*co; i++){
                 pfr.push_back(p[i]); // Copia del segmento a analizar
            }
            aux=segmentacion(pfr, nombre, ft, progresion, Acordes, mode,
                                        tc, tiempos, tonset, 0, co);
             //En aux se almacena el instante del último cambio armónico
            if(co==1){
                for(int j=0; j<(int)p.size(); j++){ /*Calculo de la
                    posición del vector general a la que corresponde el
                    último cambio armónico*/
                    if(p[j].t==aux) cont=j;
                }
            }
            pfr.clear(); //Se limpia pfr
        }else{//Caso particular: último trozo del segmento
            for(int i=cont; i<(int)p.size(); i++) pfr.push_back(p[i]);
                                    // Copia del segmento a analizar
            aux=segmentacion(pfr, nombre, ft, progresion, Acordes, mode,
                                        tc, tiempos, tonset, 1, co);
            //En aux almacenamos el instante del último cambio armónico
            aux=p[p.size()-1].t+10; pfr.clear();
                                    //Condición para salir de while
        }
    }
    tg=histograma(tc, tonset, 1); //Calculo del tempo general
    float t=tempo2(tiempos); //Calculo del tempo que más se repite
    float tb = beat(tg, tonset, fb); //Calculo de los beat y de su tempo
    escriturafichero(tc, tg, t, tb, progresion, f, ft);
    ft.close();f.close(); fb.close();
}
```

```
/*FUNCIÓN MAIN: recibir la información del exterior*/
int
main(int argc, char *argv[]){
        if (argc>5 || argc<4)
            cout << "Sintaxis: ./armonias entrada.txt prefijosalida.txt;
            cout << tablaacordes.txt valorcuantizacion" <<endl;
        else{
            if (argc==5){
                 principal (argv[1], argv[2], argv[3], atof(argv[4]));
            }
            if (argc==4){
                 principal (argv[1], argv[2], argv[3], 0);
            }
        }
        return 0;
}
```

9.2. Detección de tonalidad.

```
/*PROYECTO FINAL DE CARRERA:

TONALIDAD

        GABRIEL MESEGUER BROCAL*/

#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <fstream>
#include <vector>
#include <string.h>

using namespace std;
const string chord[]=
{"C","C#","D","Eb","E","F","F#","G","G#","A","Bb","B"};

/*Ordena el histograma de mayor a menor*/
bool operator< (const pair<string, float>  & left, const  pair<string,
float>  & right){
    bool salida=false;
    if (left.second<right.second) salida=true;
    return salida;
}

/*FUNCIÓN CAMBIO: se encarga de cambiar el nombre de los acordes por su
correspondiente valor entero: C=0; C#=1; D=2; Eb=3; E=4; F=5;
F#=6; G=7; Ab=8; A=9; Bb=10; B=11*/
void
cambio(string a, int &n){
    for (int i=0; i<12; i++){
        if (a[0]==chord[i][0] && chord[i][1]!='#' && chord[i][1]!='b')
n=i;
        if (a[0]==chord[i][0] && a[1]==chord[i][1]) n=i;
    }
}
```

```
/*FUNCIÓN TONALIDAD: calcula la tonalidad de la canción buscando
la cadencia perfecta más repetida*/

string
tonalidad(vector< pair<string, float> > &p){
    vector< pair<string, float> > h;
    vector< pair<string, float> > cont;
    float c=0;
    string result;
    for (int i=1; i<p.size(); i++){ /* Se recorre todos los cambios
                                       armónicos*/
        int n1, n2;
        cambio(p[i].first, n1); /*Se traduce cada nombre del acorde al
                                  número entero que le corresponde*/
        cambio(p[i-1].first, n2); /*Se traduce cada nombre del acorde al
                                    número entero que le corresponde*/
        int d1=n2-n1; /*Se calcula la diferencia entre cada par de
                acordes (n° de semitonos en los que se diferencian)*/
        /* Si le separan 7 semitonos se trata de una cadencia perfecta.
    Al tratarse de una representación en base 12 el opuesto a -5 es 7.
 Solo tendremos en contabilizará cuando se llegue a un acorde PM y Pm*/
        if((d1==7 || d1==-5) && (p[i].first[1]=='M' ||
        p[i].first[2]=='M' || p[i].first[1]=='m' || p[i].first[2]=='m')){
            float d = (p[i].second + p[i-1].second);
            cont.push_back(make_pair(p[i].first, d)); /*Se almacena el
                                  acorde base y el tempo que se repite*/
        }
    }

    /*Si no se encuentra ninguna cadencia Perfecta a acordes mayores o
menores se prueban con todos traduciendo el resultado a Mayor o menor.
Todos los acordes distintos de mayor y menor se traducirán en mayores a
excepción de los que estén precedidos de un acorde menor o uno
disminuido.*/
    if(cont.size()==0){
        for (int i=1; i<p.size(); i++){
            int n1, n2;
            cambio(p[i].first, n1); /*Se traduce cada nombre del acorde
                                      al número entero que le corresponde*/
            cambio(p[i-1].first, n2); /*Se traduce cada nombre del acorde
                                  al número entero que le corresponde */
            int d1=n2-n1; /*Se calcula la diferencia entre cada par de
                    acordes (n° de semitonos en los que se diferencian)*/
            if(d1==7 || d1==-5){
                string a = p[i].first;
                /*Comprobación de acordes naturales distintos de los
                                  mayores y los menores*/
```

```
        if(p[i].first[1]=='7' || p[i].first[1]=='A'
                || p[i].first[1]=='D' || p[i].first[1]=='4'
                                        || p[i].first[1]=='2'){
                    string s; s = p[i].first[0];
                    if(p[i-1].first[1]=='m' ||
                    p[i-1].first[2]=='m' || (p[i-1].first[1]=='D'
                    && p[i-1].first[2]=='i') || (p[i-1].first[2]=='D'
                    && p[i-1].first[3]=='i')) a = s + "m";
                    else a = s + "M";
                }
                /*Comprobación de acordes alterados distintos de los
                    mayores y los menores*/
                if(p[i].first[2]=='7' || p[i].first[2]=='A' ||
                    (p[i].first[2]=='D' && p[i].first[3]=='i')  ||
                            p[i].first[2]=='4'  || p[i].first[2]=='2'){
                    string s , s1;
                    s = p[i].first[0], s1 = p[i].first[1];
                    if(p[i-1].first[1]=='m' ||
                    p[i-1].first[2]=='m' || (p[i-1].first[1]=='D'
                    && p[i-1].first[2]=='i') || (p[i-1].first[2]=='D'
                    && p[i-1].first[3]=='i')) a = s + s1 + "m";
                    else a = s + s1 + "M";
                }
                float d = (p[i].second + p[i-1].second);
                cont.push_back(make_pair(a, d)); /*Se almacena el acorde
                                        base y el tempo que se repite*/
        }
    }
}
sort(cont.begin(), cont.end());
//Agrupamos todos los acordes iguales
while((int)cont.size()!=0){
    int i=0;
    float r=0;
    string aux12=cont[0].first;
    while (i<(int)cont.size()){/*Buscamos repeticiones que se repiten
                                    y las eliminamos del vector dur*/
        if (cont[i].first==aux12){
            r=r+cont[i].second;
            cont.erase(cont.begin()+i);/*Eliminamos las duraciones
                                                que se repitan*/
        }else{
            /*Los acordes que tengan los dos modos (Mayor y menor) se
            descontará al modo de mayor peso el de menor*/
            if (cont[i].first[0]==aux12[0] &&
                                cont[i].first[1]!=aux12[1]){
```

```
                    r=r-cont[i].second; cont.erase(cont.begin()+i);
                }else i++;//Contardor que permite recorrer el vector dur
            }
        }
        h.push_back(make_pair(aux12,r)); /*Se almacena el acorde base y
                                                el tempo que se repite*/
    }
    for (int o=0; o<(int)h.size(); o++){ /*Se busca base de la cadencia
                                    perfecta mayor que más se repita*/
        if(h[o].second>c){
            c=h[o].second;
            result=h[o].first;
        }
    }
    return (result);
}

/*FUNCIÓN LEERFICHERO: se encarga de la lectura del fichero de texto
entrada*/
void
leerfichero(char v[], vector< pair<string, float> > &p){
    fstream f;
    vector< pair<string, float> > aux;
    f.open(v, ios::in);
    if(f.is_open()){
        string name;
        float to;
        f>>to; //Instante de tiempo donde se produce el cambio armónico
        while(!f.eof()){
            f>>name; // tipo de cambio armónico
            aux.push_back(make_pair(name, to));
            f>>to;
        }
        aux.push_back(make_pair("  ", to)); /*El nº de cambios armónicos
siempre es una unidad menor que los tempos en los que se produce (cada
par de tiempos forman una unidad armónica con su correspondiente valor)
por eso alamacenamos el último instante de tiempo sin ningún acorde*/
        f.close();
    }else{
        cout<< "Error: Fichero no encontrado"<<endl;
    }
    for(int i=1; i<aux.size(); i++){//Calculo de las duraciones armónicas
        float dur= aux[i].second-aux[i-1].second;
        p.push_back(make_pair(aux[i-1].first, dur)); // Vector final que
contiene todos los cambios armónicos y su duración
    }
}
```

```
/*1° MAIN: Función principal: Organiza la llamada a las funciones*/
int
main(int argc, char *argv[]){
        vector< pair<string, float> > p;
        string t;
        if (argc!=2) cout << "Sintaxis: ./tonalidad entrada.txt" <<
endl;
        else{
            leerfichero(argv[1], p);
            t=tonalidad(p);
            cout<<t<<endl;
        }
        return 0;
}
```