Máster Universitario en

Ingeniería Computacional y Sistemas Inteligentes

Universidad del País Vasco Euskal Herriko Unibertsitatea

Konputazio Zientziak eta Adimen Artifiziala Saila –
Departamento de Ciencias de la Computación e Inteligencia Artificial

K I S A

I C S I

Tesis de Máster

Comparación de métodos para calcular similitud de melodías

Ander Sarriegi Gutierrez

Tutor(a/es)

Darrell Conklin
Departamento de Ciencia de la Computación e Inteligencia Artificial
Facultad de Informática

informatika fakultatea facultad de informática

KZAA /CCIA

Abril 2016

Agradecimientos

A Xenpelar Dokumentazio Zentroa por facilitar los archivos MIDI y las partituras de los bertsos para poder realizar este trabajo.

A Izaro Goienetxea Urkizu por la ayuda prestada a la hora de realizar esta tesis de máster, ayudando a recopilar información y artículos sobre el tema, a escoger una colección de melodías adecuada y a completar algunas partes de la tesis.

Resumen

En este trabajo comparamos la eficacia de diferentes métodos de comparación de melodías. Para ello, utilizamos 150 melodías de bertsos, que clasificamos en 50 familias para comparar los resultados que obtenemos de distintos algoritmos. El trabajo está estructurado de la siguiente manera: en primer lugar hacemos un estudio de los trabajos realizados por varios autores en este área para tener conocimiento de cómo podemos enfocar este problema; después presentamos teóricamente los métodos y algoritmos que hemos implementado; a continuación presentamos los resultados que hemos obtenido utilizando los métodos antes mencionados; para finalizar, analizamos los resultados obtenidos y explicamos a qué pueden deberse las diferencias entre éstos.

El objetivo del proyecto es encontrar un buen método de comparación de melodías para impulsar el uso de melodías de bertsos que apenas se utilizan, buscando melodías que son parecidas entre sí para que los bertsolaris tengan un repertorio más variado.

Índice general

1

Introducción

Las melodías que se utilizan a la hora de cantar bertsos son un elemento importante de los mismos, y se ha observado que la variedad de melodías que se usan habitualmente no es muy amplia aunque haya un gran catálogo disponible. El objetivo del proyecto es poder crear un sistema que dada una melodía, sugiera melodías parecidas, de características similares, para que sea usada por los bertsolaris, impulsando así el uso de más melodías. Para lograrlo, Xenpelar Dokumentazio Zentroa quiere implementar en el Bertso Doinutegia dicho sistema para que cuando se busque una melodía concreta, se sugieran melodías parecidas a ésta. Xenpelar Dokumentazio Zentroa es un centro de documentación creado en 1991 con el objetivo de impulsar investigaciones recogiendo, organizando y difundiendo el patrimonio de los bertsos.

Para poder crear un sistema como el mencionado, es necesario hacer una comparación entre melodías. A la hora de hacer esta comparación no podemos olvidar que las melodías para los bertsos tienen que cumplir algunas propiedades. La más importante es, posiblemente, la medida, ya que esta rige cómo debe ser la longitud de sílabas de las estrofas cantadas. Sin embargo, como la medida es un requisito textual (aunque la música acompaña al texto), y los bertsos son improvisados, la medida no es una característica muy adecuada para hacer la comparación entre melodías. Además, la medida del bertso no es una restricción para la música que lo acompaña, ya que, si es necesario, pueden cantarse varias notas para una sílaba, o pueden cantarse varias sílabas sobre la misma nota.

Bertso Doinutegia es una colección de 1987 melodías de bertsos creada por Joanito Dorronsoro y publicada por primera vez en 1995. Xenpelar Dokumentazio Zentroa renueva esta colección cada año, incluyendo nuevas melodías utilizadas tanto en competiciones como exhibiciones de bertsolaris. Las entradas en la colección tienen el nombre de la melodía, el nombre o tipo de estrofa, el género de la melodía, el creador o bertsolari que la ha utilizado, el nombre de la persona que ha añadido la melodía a la colección y el año en el que ha sido añadida. Las melodías están clasificadas en 17 tipos o géneros distintos, y esta clasificación se basa solamente en el contenido melódico.

Bertsolaritza es el arte de cantar canciones improvisadas en Euskera (bertsos) respetando varios patrones melódicos y rítmicos. En el libro "The Art of Bertsolaritza: Improvised Basque Verse Singing", Garzia et al. (2001) definen el bertso como un discurso medido, rimado y cantado. Los principales aspectos técnicos de los bertsos son la rima, la métrica y la melodía, la cuál puede clasificarse en tres grupos:

- Melodías folkloricas tradicionales.
- Nuevas arias que coinciden con la métrica tradicional.
- Melodías compuestas especificamente.

Los expertos dicen que la melodía seleccionada para cantar el bertso y la manera en la que es cantado pueden ser la llave del éxito comunicativo del bertsolari.

En este trabajo nos centramos en el contenido musical para hacer la comparación entre melodías, y compararemos la similitud entre éstas utilizando sus notas. Para ello, representaremos las notas de cada melodía como cadenas de caracteres, utilizando distintas representaciones para aplicar distintos algoritmos. Por un lado, representaremos las melodías como cadenas de notas; por otro lado, representaremos las melodías con cadenas de intervalos entre notas. Además de utilizar distintas representaciones de la melodía, utilizaremos distintos métodos de comparar las cadenas de caracteres.

Por otra parte, veremos si es mejor añadir o quitar características de la melodía a la hora de hacer el cálculo de la similitud, es decir, compararemos los resultados de calcular la similitud tomando todas las notas con los resultados de realizar este cálculo haciendo una reducción de la melodía, o haremos una comparación entre los resultados obtenidos al tener en cuenta las notas sin incluir la duración de las mismas y los resultados obtenidos incluyendo la duración de las notas.

El problema que tratamos en este trabajo está comprendido dentro del área denominada MIR ("Music Information Retrieval"). MIR es un área interdisciplinar, que involucra a investigadores de disciplinas como la musicología, ciencia computacional, ciencia de la información, ciencia cognitiva, etc. Hay varias definiciones de MIR, pero nos hemos quedado con dos de ellas:

"MIR is a multidisciplinary research endeavor that strives to develop innovative content-based searching schemes, novel interfaces, and evolving networked delivery mechanisms in an effort to make the world's vast store of music accessible to all."

[Downie, 2004]

"MIR is concerned with the extraction, analysis, and usage of information about any kind of music entity (for example, a song or a music artist) on any representation level (for example, audio signal, symbolic MIDI representation of a piece of music, or name of a music artist)."

[Schedl, 2008]

Este área abarca un gran abanico de problemas, entre otros:

- Extracción de características.
- Medidas de similitud musical.

· Recomendación de música, generación de listas de reproducción automáticas.

· Clasificación.

Desde el año 2000, ISMIR ("International Society for Music Information Retrieval") organiza anualmente un congreso que gira en torno al MIR. El proposito de ISMIR es proporcionar un lugar de encuentro para el intercambio de noticias, ideas y resultados mediante la presentación de trabajos originales, tanto teóricos como prácticos. Además, esta conferencia sirve como foro de discusión, dispone de información, tanto introductoria como exhaustiva, de dominios específicos, y sirve de escaparate para productos actuales.

A la par que la conferencia organizada por ISMIR, IMIRSEL ("International Music Information Retrieval Systems Evaluation Laboratory") de la UIUC ("University of Illinois at Urbana-Champaign") organiza una especie de competición anual para algoritmos de MIR llamada MIREX ("Music Information Retrieval Evaluation eXchange"). Algunas de las tareas que se ha pedido resolver en años anteriores son:

· Detección de la tonalidad.

· Similitud de melodía simbólica (el problema que tratamos en este trabajo).

· Query by singing/humming.

El mundo de MIR es un mundo amplio y tiene una gran variedad de aplicaciones en la vida diaria. Por ejemplo, cuando escuchamos una canción que no conocemos el nombre y queremos saber cuál es, utilizando la aplicación para móviles Shazam, podemos saber qué canción estamos oyendo. Esta aplicación hace uso del antes mencionado query by singing. La aplicación "escucha" una parte de una canción cantada, realiza un proceso ((De Bortoli, 2009), diapositivas 22-30) para comparar la canción oída con las canciones de su base de datos, y devuelve la canción más parecida como resultado.

Otra aplicación de este campo en la vida diaria es el sistema de recomendación de canciones que utilizan programas como Spotify o páginas web como youtube, que buscan en sus bases de datos canciones que tengan características parecidas a las buscadas y reproducidas por el cliente, y recomienda su escucha al mismo.

1.1. Datos

Los datos utilizados para hacer este trabajo son 50 familias de melodías obtenidas del Bertso Doinutegia presentado anteriormente. Las familias están compuestas por variaciones de una melodía, donde cada familia contiene entre 2 y 7 melodías sumando un total de 150. En el siguiente histograma podemos ver la distribución de las familias de melodías dependiendo del número de melodías que contienen.

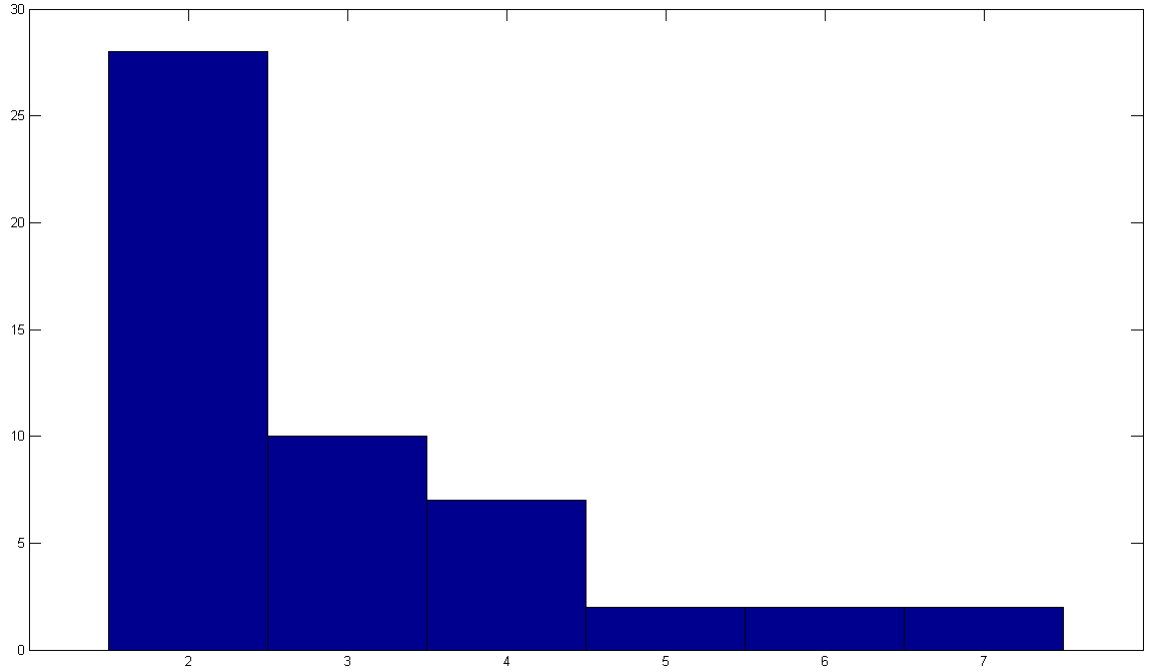


Figura 1.1: Histograma de la distribución de las familias dependiendo del número de canciones que contienen

Las melodías tienen una longitud de entre 26 y 108 notas, con una media de $53,22$ notas. Algunas de estas melodías tienen repeticiones, tanto de toda la melodía como de una sola frase. Además, todas las melodías con las que se trabaja son monofónicas, es decir, de una sola voz, exceptuando dos que tienen dos voces, en las que una voz está una tercera por debajo de la otra (casi durante toda la melodía). En estos dos casos, nos quedamos con la voz superior. Para poder extraer la información que necesitamos de las melodías para realizar el trabajo, hemos obtenido los archivos MIDI de éstas.

MIDI ("Musical Instrument Digital Interface") es un protocolo, un lenguaje, que surgió para permitir la comunicación entre sintetizadores, ordenadores, cajas de ritmo, etc. Su origen en 1983 surge de la necesidad de encontrar un sistema que posibilitase la intercomunicación de todos los aparatos que las distintas marcas iban creando, cada una con su propio lenguaje. Con la llegada del ordenador al mundo de la música pudo guardarse en los denominados archivos MIDI. Estos archivos no guardan la información sonora de manera exacta (no guardan las ondas de sonido como por ejemplo los archivos WAV), sino que la guardan de manera discreta. La gran ventaja de los archivos MIDI con respecto a otros archivos audio es que pueden editarse hasta el más mínimo detalle.

1.2. Estado del arte

En cuanto a métodos de similitud de melodías, se han realizado muchos trabajos con distintos puntos de vista para la resolución del problema. Sin embargo, en muchos de los trabajos mencionados, el principal problema por el que se implementan métodos de similitud de melodías no es el que nosotros tenemos entre manos, sino que es el denominado "query-by-humming",

es decir, tratar de encontrar la canción más parecida a la que tararea una persona en una base de datos. Aunque el objetivo del trabajo no sea el mismo, podemos aprovechar los métodos diseñados por otros, ya que en ambos casos hace falta comparar melodías para calcular la similitud entre las mismas. A la hora de realizar el cálculo de la similitud entre melodías, hay quienes han optado por un método gráfico (Urbano, 2014; Zhu et al., 2002), representando la melodía en un plano y haciendo el cálculo de la distancia con las representaciones gráficas; hay quien ha optado por representar las melodías como cadenas de distinta manera para hacer la comparación entre cadenas (Grachten et al., 2004); también hay quien ha hecho un estudio más analítico (Hu et al., 2002; Janssen et al., 2015).

En primer lugar, antes de comenzar a mencionar métodos de cálculo de similitud de melodías, es importante saber qué características debe tener cada uno de estos métodos. En uno de sus artículos, Urbano et al. (2011) nos explican los requisitos que debe cumplir un método de comparación de melodías, tanto en lo que a notas (requisitos verticales) como al tempo (requisitos horizontales) se refiere. En cuanto a requisitos verticales, explican que un método de comparación de melodías debe aceptar como iguales melodías que solo difieran en la octava en la que han sido escritas, o que estando escritas en distintas tonalidades tengan las mismas notas, o que cambiando de tonalidad, las notas sean del mismo grado en su tonalidad respectiva. Cuando un método cumple estos tres requisitos, se dice que es invariante a transposición. Estos son los requisitos en los que más nos fijaremos.

Métodos gráficos

En un estudio para MIREX (Music Information Retrieval Evaluation eXchange) Urbano (2014) presenta un método que ha ido perfeccionando durante los últimos años, obteniendo el primer puesto en todas las medidas de efectividad en 2014. En primer lugar, representa las notas de la melodía monofónica (de una sola voz) en un plano nota/tiempo, donde el valor nota se representa mediante intervalos dirigidos, y el valor tiempo es el momento en el que comienza cada nota. Una vez representadas todas las notas, se calcula mediante interpolación la curva que pasa por todos los puntos. Una vez obtenidas las curvas, emplea tres métodos de comparación que denomina: "ShapeH", "Time" y "ShapeTime".

Otros que trabajan con una representación gráfica de la melodía son Zhu et al. (2002). Éstos representan la melodía mediante el "contorno continuo de la melodía" en un plano nota/tiempo, hacen alineamiento de melodías usando pendientes y presentan un método para calcular similitud entre contornos. Para crear el contorno continuo de la melodía de un archivo MIDI, primero se extraen las notas y cada una de ellas es convertida en un segmento de línea horizontal, de manera que la altura la da la nota y la longitud la da la duración de la misma. En caso de que haya dos o más notas iguales consecutivas, la longitud de la línea es igual a la duración total de todas esas notas, y si hay un silencio, la duración del silencio se añade a la de la última nota. El resultado es una secuencia de segmentos de línea. Una vez obtenido el contorno, cogiendo los segmentos correspondientes a los máximos y mínimos locales (picos y valles) y uniendo el principio de uno de ellos con el principio del siguiente opuesto, obtienen las pendientes. Utilizando estas pendientes se define un método de alineamiento. Primero se hace el alineamiento horizontal, y luego el vertical. Una vez hecho el alineamiento, calculan la similitud entre las pendientes de las melodías. Los mismos autores han publicado dos artículos en los que se explica más a fondo el método de alineamiento y el cálculo de similitud (Zhu

et al., 2001a,b).

Aloupis et al. (2006) cogen la idea presentada por Ó Maidín (1998), que propuso una distancia geométrica entre dos melodías, que eran modeladas como funciones rectilíneas de nota-duración. Después, medía la diferencia entre ambas melodías como el área mínima entre las dos cadenas poligonales, permitiendo movimientos verticales de dichas cadenas. Los autores de este artículo trabajan con melodías cíclicas, que representan como cadenas poligonales ortogonales en la superficie de un cilindro, y presentan dos algoritmos para calcular el área mínima de dos melodías ortogonales. Las melodías cíclicas son aquellas que repiten un dibujo una y otra vez

Añadiendo una característica más que los anteriores autores a la hora de representar las melodías en un plano, Typke et al. (2003) utilizan distancias de transporte ("Earth Mover's Distance" y "Proportional Transportation Distance") sobre una representación de las notas como puntos con peso asignado en un plano nota/tiempo, haciendo un ajuste de dichas representaciones en el momento que van a comparar dos melodías. Por otra parte, Orio y Rodà (2009) representan una colección de partituras utilizando una estructura de grafo, donde los nodos terminales describen el contenido musical, los nodos internos representan su generalización gradual y los arcos denotan la relación entre ellos. La similitud entre dos melodías es calculada analizando la estructura de grafo y buscando el camino más corto entre los nodos correspondientes dentro del grafo.

Métodos analíticos

Grachten, Arcos y López de Mántaras (2004) presentan el modelo "Implication/Realization" (I/R) de Narmour (1990; 1992), y comparan cuatro distancias basadas en "Edit-Distance" para diferentes "niveles": secuencias de notas, secuencias de contorno (con relación entre intervalos y sin relación entre intervalos), y secuencias I/R. El método de comparación en secuencias de notas presentado en el último artículo mencionado lo presentaron Mongeau y Sankoff (1990), que añadían dos operaciones más al Edit-Distance: consolidación y fragmentación. La consolidación consiste en reemplazar varios elementos por uno solo, y la fragmentación consiste en reemplazar un elemento por varios. Por lo tanto, tenían un método para calcular la distancia que constaba de cinco operaciones: reemplazar, eliminar, insertar, consolidar y fragmentar, y para el cálculo de cada una de las operaciones tenían en cuenta la duración y la nota del caracter de la secuencia.

Hu et al. (2002) presentan un modelo probabilístico para calcular similitud entre melodías, donde estiman cuál es la probabilidad de que una melodía derive de otra. Explican cómo utilizar esta aproximación para buscar bases de datos de melodías y comparan los resultados obtenidos con los resultados del Edit-Distance.

En su artículo Janssen et al. (2015) comparan los resultados obtenidos al implementar distintos métodos para calcular la similitud de melodías utilizando métodos parecidos a los que vamos a utilizar en este trabajo utilizando fragmentos de melodías en lugar de melodías completas, y se realiza el trabajo buscando frases similares. Para evaluar sus resultados, los comparan con el análisis realizado por tres expertos. Los autores realizan el trabajo con una colección de 360 canciones folkloricas alemanas, que dividen en 26 familias de la misma manera que lo hacemos en este trabajo, es decir, las canciones que son variaciones de una misma pertenecen a la misma

familia. Müllensiefen y Frieler (2004) no se limitan a implementar y comparar distintos métodos, sino que hacen una combinación de distintos métodos. En su artículo implementan unas 50 medidas de similitud de melodías que difieren en la manera de transformar los datos musicales y en los algoritmos computacionales. Tras un experimento en el que tres oyentes comparaban el rendimiento de las diferentes medidas de similitud con clasificaciones de humanos expertos, obtienen un modelo optimizado usando regresión lineal, que combina la salida de diversas medidas representando diferentes dimensiones musicales.

El los artículos mencionados hasta ahora, los valores para calcular la similitud entre melodías podían obtenerse mediante archivos MIDI. Ahora, vamos a mencionar un artículo que trabaja con archivos audio en formato WAV. En este artículo, Flexer et al. (2006) presentan una combinación probabilística entre diferentes características de la melodía, como son la similitud espectral y la similitud rítmica. Para calcular la similitud espectral siguen los siguientes pasos: primero, para cada canción, se calculan los MFCC ("Mel Cepstrum Central Coefficients") para frames cortos superpuestos, luego se calcula un GMM ("Gaussian Mixture Model") para cada canción, y para finalizar, se calcula la matriz de distancias entre las canciones utilizando la probabilidad de cada canción dado un GMM. Para calcular la similitud rítmica, utilizan un algoritmo presentado a la competición MIREX 2005 en "Perceptual tempo induction", denominado como *Algorithm1* en [1] (Gouyon y Dixon, 2005). Una vez calculadas estas similitudes, las combinan utilizando distintos clasificadores.

En cuanto a identificar automáticamente a qué familia pertenecen las canciones Savage y Atkinson (2015) hacen un estudio, utilizando siempre el mismo método de alineamiento, para ver con qué parámetros de la función de alineamiento y características de la canción se obtienen mejores resultados. Sin embargo, entendemos que las familias de canciones con las que trabajan ellos son tipos de canciones distintas, mientras que las "familias" que tenemos entre manos en este trabajo son variaciones de una misma canción, siendo todas ellas bertsos, por lo que creemos que el método implementado no funcionará bien en nuestro caso.

Evaluación de resultados

Una vez calculados los resultados de los métodos anteriores, es importante evaluar los resultados de dichos métodos. El parecido de las canciones es un tema subjetivo, ya que dos personas distintas pueden diferir en la opinión al analizar el parecido entre dos canciones. Sin embargo, existen métodos para evaluar de manera científica los resultados devueltos por los métodos de comparación de melodías. En los artículos mencionados hasta ahora, se implementan diferentes métodos para hacer esto. Janssen et al. (2015) comparan los resultados de los algoritmos con las opiniones dadas por humanos. Para distintos valores de threshold, se miran las coincidencias de los algoritmos con los resultados de los humanos, es decir, los positivos reales y los falsos positivos, y la relación entre éstos se representa mediante la curva ROC utilizando el threshold como parámetro. El área bajo la curva determina cuál de los algoritmos funciona mejor. Müllensiefen y Frieler (2004) también utilizan este método para evaluar los resultados de sus algoritmos.

Orio y Rodà (2009) utilizan tres medidas de efectividad llamadas "Average Precision", "R-

[1] *"http://mtg.upf.edu//files/publications/22b3bb-ISMIR-MIREX05-Gouyon.pdf"*

Precision", como medidas de efictividad corrientes, y "Average Dynamic Recall", que tiene en cuenta que los juicios de relevancia no son binarios. Por otra parte, Savage y Atkinson (2015) utilizan el test de correlación de de matriz de distancias de Mantel para evaluar el algoritmo de alineamiento de secuencias, y una relación entre positivos reales y falsos positivos en un método utilizado previamente por van Kranenburg et al. (2013) que define el valor J de la siguiente manera:

$$J = \frac{tpr}{1 + fpr}$$

donde *tpr* es el ratio de positivos reales y *fpr* es el ratio de falsos positivos.

Otro método para comparar resultados es el llamado F_1-*score* (también *F-score* o *F-measure*), que puede ser interpretada como una media ponderada de Precision y Recall, donde alcanza su mejor valor en 1 y el peor en 0. El valor de F_1-*score* balanceado se calcula mediante la media harmónica de Precision y Recall, es decir:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

En este trabajo, el método empleado para comparar los resultados de los distintos algoritmos implementados el método Precision-Recall, calculando los vectores correspondientes a cada algoritmo y comparando las representaciones gráficas de estos vectores. En la sección 2,4 explicamos el funcionamiento de este método.

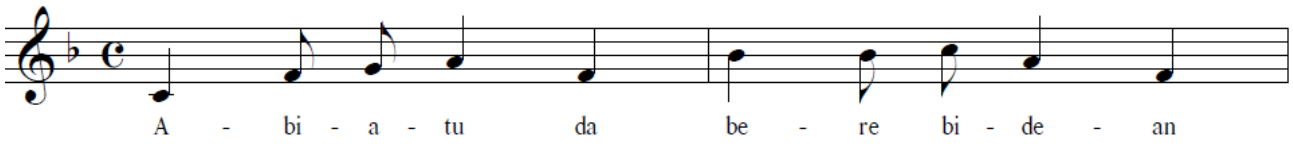
2

Métodos

En esta sección se explican todos los métodos empleados en el trabajo. En primer lugar, explicaremos brevemente cómo hemos leído los datos y cómo los hemos representado. En segundo lugar, explicaremos los métodos de alineamiento utilizados, que son los métodos con los que hemos calculado la similitud o distancia entre las melodías. A continuación, explicaremos un método de reducción de melodías que hemos implementado. Para finalizar, hablaremos del método empleado para comparar los resultados obtenidos. La melodía utilizada como ejemplo para explicar cómo funcionan los métodos que tenemos a continuación es *"Abiatu da bere bidean"*.

2.1. Lectura de datos

Como hemos mencionado anteriormente, los archivos con los que trabajamos son archivos MIDI, los cuales guardan toda la información de las melodías de forma simbólica. Para poder calcular la similitud entre melodías, necesitamos extraer la información necesaria de dichos archivos. Para ello, utilizaremos el código de lectura para archivos MIDI que hemos encontrado en [1], el cuál hemos adaptado para obtener la información que queremos. Cuando leemos un archivo MIDI con este código, obtenemos el siguiente output (enseñamos las líneas que nos interesan, ya que obtenemos más lineas que muestran otros mensajes que no nos aportan información, que son un fragmento de la melodía, y la partitura correspondiente a esa melodía).



[1] *"http://stackoverflow.com/questions/3850688/reading-midi-files-in-java"*

@0	Channel: 0 Note on,	C4	key=60 velocity: 64
@1024	Channel: 0 Note off,	C4	key=60 velocity: 0
@1024	Channel: 0 Note on,	F4	key=65 velocity: 64
@1536	Channel: 0 Note off,	F4	key=65 velocity: 0
@1536	Channel: 0 Note on,	G4	key=67 velocity: 64
@2048	Channel: 0 Note off,	G4	key=67 velocity: 0
@2048	Channel: 0 Note on,	A4	key=69 velocity: 64
@3072	Channel: 0 Note off,	A4	key=69 velocity: 0
@3072	Channel: 0 Note on,	F4	key=65 velocity: 64
@4096	Channel: 0 Note off,	F4	key=65 velocity: 0
@4096	Channel: 0 Note on,	A♯4	key=70 velocity: 64
@5120	Channel: 0 Note off,	A♯4	key=70 velocity: 0
@5120	Channel: 0 Note on,	A♯4	key=70 velocity: 64
@5632	Channel: 0 Note off,	A♯4	key=70 velocity: 0
@5632	Channel: 0 Note on,	C5	key=72 velocity: 64
@6144	Channel: 0 Note off,	C5	key=72 velocity: 0
@6144	Channel: 0 Note on,	A4	key=69 velocity: 64
@7168	Channel: 0 Note off,	A4	key=69 velocity: 0
@7168	Channel: 0 Note on,	F4	key=65 velocity: 64
@8192	Channel: 0 Note off,	F4	key=65 velocity: 0

De estas líneas podemos extraer la siguiente información:

1. El símbolo @ nos indica que comienza un nuevo mensaje.
2. El número que viene a continuación nos indica en que “tick” ocurre el evento.
3. Después tenemos el canal por el que se manda el mensaje.
4. Lo siguiente que tenemos es la indicación de si la nota comienza o acaba.
5. A continuación, tenemos la nota y la octava en la que está y el valor que corresponde a esa nota en los archivos MIDI.
6. Para finalizar tenemos la velocidad.

La información que nos interesa a nosotros es el número que indica el “tick” en el que ocurre el evento y el valor numérico de las notas, ya que de los primeros valores podemos sacar la duración de las notas, y utilizando los segundos sabemos qué nota tenemos.

@0	Channel: 0 Note on,	C4	key=60 velocity: 64
@1024	Channel: 0 Note off,	C4	key=60 velocity: 0
@1024	Channel: 0 Note on,	F4	key=65 velocity: 64
@1536	Channel: 0 Note off,	F4	key=65 velocity: 0
@1536	Channel: 0 Note on,	G4	key=67 velocity: 64
@2048	Channel: 0 Note off,	G4	key=67 velocity: 0
@2048	Channel: 0 Note on,	A4	key=69 velocity: 64
@3072	Channel: 0 Note off,	A4	key=69 velocity: 0
@3072	Channel: 0 Note on,	F4	key=65 velocity: 64
@4096	Channel: 0 Note off,	F4	key=65 velocity: 0
@4096	Channel: 0 Note on,	A♯4	key=70 velocity: 64
@5120	Channel: 0 Note off,	A♯4	key=70 velocity: 0
@5120	Channel: 0 Note on,	A♯4	key=70 velocity: 64
@5632	Channel: 0 Note off,	A♯4	key=70 velocity: 0
@5632	Channel: 0 Note on,	C5	key=72 velocity: 64
@6144	Channel: 0 Note off,	C5	key=72 velocity: 0
@6144	Channel: 0 Note on,	A4	key=69 velocity: 64
@7168	Channel: 0 Note off,	A4	key=69 velocity: 0
@7168	Channel: 0 Note on,	F4	key=65 velocity: 64
@8192	Channel: 0 Note off,	F4	key=65 velocity: 0

Para obtener la duración de las notas, es necesario hacer la resta entre los valores de "tick" del mensaje "Note off" y del mensaje "Note on" de la misma nota. En nuestro caso, esta operación es sencilla, ya que solo nos quedamos con estos mensajes, y al trabajar con melodías monofónicas, una nota no comienza hasta que termina la anterior, por lo tanto, nos sirve coger las líneas de dos en dos y hacer la resta entre los valores de estas dos líneas.

@0	Channel: 0 Note on,	C4	key=60 velocity: 64
@1024	Channel: 0 Note off,	C4	key=60 velocity: 0
@1024	Channel: 0 Note on,	F4	key=65 velocity: 64
@1536	Channel: 0 Note off,	F4	key=65 velocity: 0
@1536	Channel: 0 Note on,	G4	key=67 velocity: 64
@2048	Channel: 0 Note off,	G4	key=67 velocity: 0
@2048	Channel: 0 Note on,	A4	key=69 velocity: 64
@3072	Channel: 0 Note off,	A4	key=69 velocity: 0
@3072	Channel: 0 Note on,	F4	key=65 velocity: 64
@4096	Channel: 0 Note off,	F4	key=65 velocity: 0
@4096	Channel: 0 Note on,	A♯4	key=70 velocity: 64
@5120	Channel: 0 Note off,	A♯4	key=70 velocity: 0
@5120	Channel: 0 Note on,	A♯4	key=70 velocity: 64
@5632	Channel: 0 Note off,	A♯4	key=70 velocity: 0
@5632	Channel: 0 Note on,	C5	key=72 velocity: 64
@6144	Channel: 0 Note off,	C5	key=72 velocity: 0
@6144	Channel: 0 Note on,	A4	key=69 velocity: 64
@7168	Channel: 0 Note off,	A4	key=69 velocity: 0
@7168	Channel: 0 Note on,	F4	key=65 velocity: 64
@8192	Channel: 0 Note off,	F4	key=65 velocity: 0

El problema que tenemos al leer los datos de esta manera es que los silencios de las melodías

no son identificados de manera automática. Para poder identificar los silencios, es necesario comparar los valores de "tick" entre el comienzo de una nota y el final de la anterior. En caso de que ambos valores sean distintos, se puede suponer que entre esas dos notas hay un silencio. Sin embargo, puede que esa diferencia se deba a que hay varios mensajes de otro tipo entre esas dos notas, por lo que no es sencillo identificar los silencios.

2.2. Representación de datos

Puesto que en este trabajo se comparan tanto distintos algoritmos para calcular similitud entre melodías, como los resultados obtenidos utilizando diferentes propiedades de las melodías, es necesario representar éstas de diferentes maneras. En este trabajo utilizaremos tres representaciones distintas de cada melodía: en la primera representación, las melodías serán representadas mediante cadenas de caracteres que tendrán en cuenta las notas, pero no su duración; en la segunda, las melodías serán representadas utilizando los intervalos diatónicos entre las notas; y en la tercera, se representarán con cadenas de caracteres que tendrán en cuenta tanto la nota como su duración.

Para la primera representación, la más sencilla de las tres, nos basta con quedarnos con el valor numérico de cada nota una vez transportada la melodía a Do Mayor. Utilizaremos la representación de las notas en caracteres propuesta por Frey (2008). Sin embargo, ante la dificultad ya mencionada de identificar los silencios, no añadiremos estos a la representación, por lo tanto, no necesitaremos el caracter R propuesto por Frey para los silencios. La representación de la melodía "Abiatu da bere bidean" de esta forma sería la siguiente:

$$DGABGCCDBGCCDBCBAGDGABGCCDBGCCDBAGEDEXGAAGABCCXGAB$$
$$GABCDDCBABDDCBABDDCBCDEDGABCCXGABGGABCBAGXGABCBAAG$$

Si aplicamos el método de reducción descrito a esta melodía y utilizamos esta misma representación, la melodía quedaría de la siguiente manera:

$$DBGCCDGCCDBCDBGCCDGCCDDAAGC$$
$$CXBGDDADDADDBEGCCXBGGCXCAAG$$

La segunda representación requiere de la realización de algunos cálculos sencillos. En primer lugar, al no representar las notas, sino los intervalos diatónicos entre estas, tenemos que hacer la resta entre los valores de nota de cada nota con la anterior, y después mirar cuántos tonos enteros de diferencia tenemos. Como los valores de las notas aumentan en un valor por cada semitono que subimos y disminuyen uno por cada semitono que bajamos, y estamos calculando cuántos tonos de diferencia hay entre las notas, haremos la siguiente operación para calcular los intervalos diatónicos:

a) Si el resultado de la resta es positivo, dividimos ese resultado entre dos y nos quedamos con el techo (el menor número entero mayor o igual al número real) de ese valor.

b) Si la diferencia es 0, el intervalo diatónico también es 0.

c) Si el resultado de la resta es negativo, dividimos ese resultado entre dos y nos quedamos con el suelo (el mayor número entero menor o igual al número real) de ese valor.

Una vez tenemos los intervalos diatónicos calculados, cambiamos los números por caracteres siguiendo la siguiente tabla:

Cuadro 2.1: Tabla que muestra el caracter correspondiente al número del intervalo diatónico

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3	4	5	6	7	8	9	10	11	12	13

Utilizando el mismo ejemplo de siempre, esta sería la representación en intervalos diatónicos:

pnnkpmnkkpmnknlllj pnnkpmnkkpmnkllklnnnnmlnnnmjnnnk
nnnnmlllnomlllnomllnnnlinnnmjnnnkmnnnllllnnnnllml

Si representamos de esta manera la melodía reducida, obtenemos la siguiente cadena de caracteres:

rkpmnipmnknhrkpmnipmngqmlpmjpkqmjpmjpmkphpmjpkmpjpkml

Para terminar, la tercera representación es la propuesta por Frey, pero sin tener en cuenta los silencios. Para cada nota, tenemos en cuenta la duración, y ponemos el mismo caracter de la nota repetido tantas veces como sea necesario para completar la duración de dicha nota teniendo en cuenta que cada caracter representa la duración mínima que se ha fijado con anterioridad. En nuestro caso, esta duración mínima será de una semicorchea, es decir, la cuarta parte de un tiempo. De esta manera, si tenemos una nota con duración de corchea, repetiremos el mismo caracter dos veces, si tenemos una nota con duración de negra, repetiremos el caracter cuatro veces, etcétera.

Sin embargo, en nuestro conjunto de melodías hay algunas que tienen tresillos de corchea (trata de introducir tres corcheas en el tiempo de dos), las cuales no pueden dividirse en semicorcheas. Para estos casos, se transformará la primera corchea del tresillo en corchea y las otras dos en semicorcheas. De esta manera, aunque alteremos un poco la duración de algunas notas, no tendremos problemas a la hora de hacer la división en semicorcheas.

La representación de "Abiatu da bere bidean" utilizando este método es la siguiente:

DDDDGGAABBBBGGGGCCCCCCDDBBBBGGGGCCCCCCDDBBBBCCCCBBB
BAAAAGGGGDDDDGGAABBBBGGGGCCCCCCDDBBBBGGGGCCCCCCDDBB
BBAAAAGGGGEEEEDDDDEEEEXXGGAAAAAAAAGGGGAABBCCCCCCCCX
XXXGGAABBBBGGGGAAAABBBBCCCCDDDDDDCCBBAABBBBDDDDDDCC
BBAABBBBDDDDDDCCBBBBCCCCDDDDEEEEDDDDGGGGAABBCCCCCCC
CXXXXGGAABBBBGGGGGGGGAABBCCCCBBBBAAAAGGGGXXXXGGGGAA
BBCCCCBBBBAAAAAAAAGGGG

2.3. Alineamiento

En este apartado explicaremos los métodos de alineamiento utilizados en el trabajo. Estos son los métodos con los que calcularemos la similitud o la distancia entre las melodías, que es la parte más importante de este trabajo.

Algoritmo de Smith-Waterman modificado

T. F. Smith y M. S. Waterman (1981) publicaron un artículo titulado *"Identification of Common Molecular Subsequences"* donde describían un algoritmo que mejoraba los resultados de investigaciones anteriores de ambos autores. Este algoritmo mide el número mínimo de eventos requeridos para convertir una secuencia de cadena de caracteres en otra. Smith y Waterman diseñaron este algoritmo para buscar relación entre cadenas de aminoacidos, y cadenas de ADN y ARN.

El problema de alinear una cadena $A = (a_0...a_{n-1})$ de longitud n con una cadena $B = (b_0...b_{m-1})$ de longitud m, siendo $m \geqslant n$ es algo trivial. Se alinea el caracter a_0 con b_0 y se mira si coincide la cadena A con la subcadena de B correspondiente. Se va deslizando A a través de B, alineando a_0 con b_1, b_2, etc. y se mira si hay coincidencia. El algoritmo de Smith-Waterman hace lo que hemos descrito, planteando además un método para expandir y contraer la cadena A utilizando sustituciones y eliminaciones para hacerlo coincidir con la cadena B.

Como el algoritmo de Smith-Waterman es un algoritmo de programación dinámica, garantiza que encontrará el resultado óptimo de alineamiento local respecto al sistema de puntuación que sea utilizado. La puntuación de sustitución viene dada de una "matriz de sustitución", y la puntuación de abrir un hueco o dejar un vacío también es preestablecida. Para encontrar dicha solución óptima, el algoritmo crea una matriz de puntuación entre las dos secuencias que están siendo comparadas.

Frey en su tesis (2008) hace una adaptación del algoritmo de Smith-Waterman para utilizarlo con melodías. Frey representa las melodías como cadenas de caracteres, donde tiene en cuenta la nota que es y la duración de ésta, cogiendo una duración mínima como base, y repitiendo el caracter las veces que haga falta hasta darle su duración completa. Pero antes de empezar a calcular similitud entre melodías, es necesario preprocesar los datos para que los resultados obtenidos sean fiables. En primer lugar es necesario trasportar todas las melodías a Do Mayor. Para eso, nos fijamos en la armadura, y sumamos o restamos un número fijo a cada valor de nota. En el cuadro 2.2 mostramos los valores que hay que sumar o restar dependiendo la armadura (número de bemoles o sostenidos).

Cuadro 2.2: Tabla que muestra el número que hay que sumar a los valores de notas dependiendo de la tonalidad de la melodía

Do ♭ M	Sol ♭ M	Re ♭ M	La ♭ M	Mi ♭ M	Si ♭ M	Fa M	Do M	Sol M	Re M	La M	Mi M	Si M	Fa ♯ M	Do ♯ M
7♭	6♭	5♭	4♭	3♭	2♭	1♭	0	1♯	2♯	3♯	4♯	5♯	6♯	7♯
+1	−6	−1	+4	−3	+2	−5	0	−7	−2	+3	−4	+1	−6	−1

Al pensar en la tonalidad no se tiene en cuenta si la melodía está en modalidad Mayor o menor, se cogen siempre como si estuvieran en modalidad Mayor.

Una vez hemos pasado todas las melodías a Do Mayor, se representan en cadenas de caracteres. Como tenemos notas que requieren de dos caracteres (por ejemplo, Do sostenido=$C\sharp$), se hace la siguiente transformación para que el algoritmo de Smith-Waterman no tenga problemas:

$$C\sharp = D\flat = V$$
$$D\sharp = E\flat = W$$
$$F\sharp = G\flat = X$$
$$G\sharp = A\flat = Y$$
$$A\sharp = B\flat = Z$$

Como hemos mencionado antes, para utilizar el algoritmo de Smith-Waterman necesitamos una matriz de sustitución y una puntuación para los huecos abiertos. Puesto que utilizamos distintos tipos de representación, necesitamos distintas matrices de sustitución y valores para los huecos abiertos.

En el caso de la representación de notas, utilizaremos la matriz de sustitución y el valor de huecos abiertos presentados por Frey (2008). La matriz de sustitución es la siguiente:

Cuadro 2.3: Matriz de sustitución para representación de notas

	C	V	D	W	E	F	X	G	Y	A	Z	B	R
C	10	−1	−2	−3	−4	−5	−6	−5	−4	−3	−2	−1	−7
V	−1	10	−1	−2	−3	−4	−5	−6	−5	−4	−3	−2	−7
D	−2	−1	10	−1	−2	−3	−4	−5	−6	−5	−4	−3	−7
W	−3	−2	−1	10	−1	−2	−3	−4	−5	−6	−5	−4	−7
E	−4	−3	−2	−1	10	−1	−2	−3	−4	−5	−6	−5	−7
F	−5	−4	−3	−2	−1	10	−1	−2	−3	−4	−5	−6	−7
X	−6	−5	−4	−3	−2	−1	10	−1	−2	−3	−4	−5	−7
G	−5	−6	−5	−4	−3	−2	−1	10	−1	−2	−3	−4	−7
Y	−4	−5	−6	−5	−4	−3	−2	−1	10	−1	−2	−3	−7
A	−3	−4	−5	−6	−5	−4	−3	−2	−1	10	−1	−2	−7
Z	−2	−3	−4	−5	−6	−5	−4	−3	−2	−1	10	−1	−7
B	−1	−2	−3	−4	−5	−6	−5	−4	−3	−2	−1	10	−7
R	−7	−7	−7	−7	−7	−7	−7	−7	−7	−7	−7	−7	10

y el valor para los huecos abiertos es −8.

En esta matriz podemos ver que si los caracteres coinciden, reciben un valor de 10, y en caso de que los caracteres no coincidan, el valor que reciben es un valor negativo igual al menor número de semitonos que hay entre ambas notas (como la escala de notas es cíclica, podemos llegar, por ejemplo, de C a A de dos maneras, subiendo 9 semitonos o bajando 3). Puesto que el silencio, denotado en estas representaciones con el caracter R, no es ninguna nota, la diferencia

de cualquiera de las notas con el silencio es mayor a la que puede haber entre dos notas, por eso se le asigna el valor -7. En cuanto a los huecos abiertos, es un valor que puede variar, como ya se ha mencionado anteriormente, y en este caso se le da un valor menor a la mayor diferencia que hay en la matriz, que es de -7.

En el caso en el que representamos las melodías con intervalos diatónicos (ver cuadro 2.1), la matriz de sustitución utilizada es la siguiente:

Cuadro 2.4: Matriz de sustitución para representación en intervalos diatónicos

	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u
f	10	−1	−2	−3	−4	−5	−6	−7	−8	−9	−10	−11	−12	−13	−14	−15
g	−1	10	−1	−2	−3	−4	−5	−6	−7	−8	−9	−10	−11	−12	−13	−14
h	−2	−1	10	−1	−2	−3	−4	−5	−6	−7	−8	−9	−10	−11	−12	−13
i	−3	−2	−5	10	−1	−2	−3	−4	−5	−6	−7	−8	−9	−10	−11	−12
j	−4	−3	−2	−1	10	−1	−2	−3	−4	−5	−6	−7	−8	−9	−10	−11
k	−5	−4	−3	−2	−1	10	−1	−2	−3	−4	−5	−6	−7	−8	−9	−10
l	−6	−5	−4	−3	−2	−1	10	−1	−2	−3	−4	−5	−6	−7	−8	−9
m	−7	−6	−5	−4	−3	−2	−1	10	−1	−2	−3	−4	−5	−6	−7	−8
n	−8	−7	−6	−5	−4	−3	−2	−1	10	−1	−2	−3	−4	−5	−6	−7
o	−9	−8	−7	−6	−5	−4	−3	−2	−1	10	−1	−2	−3	−4	−5	−6
p	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	10	−1	−2	−3	−4	−5
q	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	10	−1	−2	−3	−4
r	−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	10	−1	−2	−3
s	−13	−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	10	−1	−2
t	−14	−13	−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	10	−1
u	−15	−14	−13	−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	10

y el valor para los huecos abiertos es -16.

Estos valores los asignamos siguiendo la misma idea con la que Frey creó su matriz de sustitución, es decir, cuando los caracteres coinciden, le damos un valor de 10, cuando no coinciden, le damos un valor negativo igual al número de de tonos de diferencia que representa cada uno de los caracteres, y le damos un valor negativo mayor que todos los demás.

Algoritmo de Needleman-Wunsch

Al igual que el algoritmo de Smith-Waterman, el algoritmo de Needleman-Wunsch es un algoritmo que se utiliza para alinear secuencias de proteínas o nucleótidos. Los creadores de este algoritmo son S. B. Needleman y C. D. Wunsch (1970). Este algoritmo fue una de las primeras aplicaciones de programación dinámica en comparación de secuencias biológicas.

El algoritmo de Needleman-Wunsch necesita también una matriz de sustitución y un valor para los huecos abiertos. Por lo tanto, utilizaremos los mismos valores que hemos utilizado en el algoritmo de Smith-Waterman. Es decir, la matriz de sustitución para representación de notas, con o sin duración, es la de la figura 2.3 y el valor de abrir huecos es de -8, y en el caso de representación de intervalos la matriz de sustitución es la de la figura 2.4, y el valor de abrir

huecos es de -16.

A diferencia del algoritmo de Smith-Waterman, añadimos un valor de extensión de huecos, que será la cuarta parte del valor para huecos abiertos:

$$\frac{\text{valor de hueco abierto}}{\text{valor de extensión de hueco}} = 4$$

Es decir, en el algoritmo de Smith-Waterman, si tenemos varios huecos seguidos, todos tienen el mismo valor, que será el valor de abrir huecos, mientras que en el algoritmo de Needleman-Wunsch, si tenemos varios huecos seguidos, el primero tendrá el valor de abrir hueco y el resto tendrá el valor de extensión de hueco.

La mayor diferencia entre los algoritmos de Smith-Waterman y de Needleman-Wunsch es que el primero es un algoritmo de alineamiento local, y el segundo es un algoritmo de alineamiento global. Por tanto, si tenemos dos melodías siendo una parte de la otra, el algoritmo de Smith-Waterman dará un valor de similitud máximo, mientras que el algoritmo de Needleman-Wunsch añade huecos a la canción corta hasta obtener la misma longitud que la canción larga.

Edit-Distance

Edit-Distance, o distancia de Levenshtein, es una distancia definida por el científico ruso Vladimir Levenshtein en 1965 como el mínimo número de operaciones de edición necesarias para transformar una cadena de caracteres en otra, considerando tres operaciones posibles: inserción, sustitución y borrado.

Cada una de esas operaciones tiene un peso, siendo el peso de la inserción y borrado de 1, y el peso de la sustitución de 0 en el caso de que ambos caracteres sean iguales y de 1 en caso de que sean diferentes.

Al igual que hemos dicho del algoritmo de Smith-Waterman, el algoritmo del Edit-Distance es un algoritmo de programación dinámica, lo que nos asegura que encontraremos un resultado óptimo local. Como las cadenas que vamos a utilizar nosotros no son extremadamente largas, podemos suponer que ese óptimo local que obtenemos es el óptimo global.

Puesto que los métodos presentados aquí calculan cosas distintas (los algoritmos de Smith-Waterman y Needleman-Wunsch miden similitud entre canciones, mientras que Edit-Distance mide distancia entre canciones), es necesario normalizar los resultados para poder compararlos.

Las normalizaciones que hemos utilizado nos dejan los resultados de estos algoritmos en el intervalo $[0, 1]$, y utilizando la formula $\sigma(i, j) = 1 - d(i, j)$, donde $d(i, j)$ es la distancia entre las canciones i y j, y $\sigma(i, j)$ es la similitud entre las canciones i y j, podemos transformar los resultados normalizados de Edit-Distance en similitud, de manera que los resultados de ambos algoritmos pueden compararse. Las normalizaciones utilizadas son las siguientes:

$$\frac{\text{Smith-Waterman}(q_1, q_2)}{min(\text{Smith-Waterman}(q_1, q_1), \text{Smith-Waterman}(q_2, q_2)) \times 10}$$

$$\frac{\text{Needleman-Wunsch}(q_1, q_2)}{min(\text{Needleman-Wunsch}(q_1, q_1), \text{Needleman-Wunsch}(q_2, q_2)) \times 10}$$

$$\frac{\text{Edit-Distance}(q_1, q_2) - |l_1 - l_2|}{min(l_1, l_2)}$$

donde l_1 y l_2 son las longitudes de las cadenas de caracteres de las canciones q_1 y q_2 respectivamente.

En el caso de los algoritmos de Smith-Waterman y Edit-Distance, aunque ya tenemos los valores devueltos por ambos algoritmos entre 0 y 1, al aplicar el método Precision-Recall podemos encontrar errores, como por ejemplo, que tengamos puntos $(0, 0)$ en los gráficos. Por eso, una vez ordenados los resultados de cada fila de mayor a menor, dividimos cada fila de la matriz con el valor máximo de esa misma fila. De esta manera, el mayor de los valores de la fila será 1 para todas las filas, evitando los posibles problemas de Precision-Recall.

En el caso del algoritmo de Needleman-Wunsch obtenemos valores positivos y negativos, por tanto, para que todos los valores estén entre 0 y 1, a cada fila de la matriz le restamos el mínimo de esa fila, y después dividimos cada fila con el valor máximo de la fila. De esta manera, todos los valores de la matriz estarán entre 0 y 1, siendo el máximo de cada fila 1 y el mínimo 0 para todas las filas. De esta forma evitamos posibles problemas con el método Precision-Recall.

2.4. Algoritmo de reducción

Cuando tenemos una colección de melodías extensa, es importante buscar una manera de poder reducir éstas de forma que, sin eliminar información importante, consigamos una representación más breve de las mismas, facilitando así tareas como la comparación de melodías. Hay varias maneras de reducir las melodías. En sus tesis, Wallentinsen (2013) y Bor (2009) hablan del método presentado por Morris (1993), que consta de dos partes: en la primera, se eliminan las notas que no son máximos o mínimos locales, y en la segunda, se eliminan las notas que, de las notas que quedan del paso anterior, no son máximos entre los máximos ni mínimos entre los mínimos. Otro algoritmo de reducción que presenta Bor es el que llama "3-window" que funciona de la siguiente manera: se coge una nota junto con la anterior y la posterior, se mira si la nota central es máximo o mínimo, y si no lo es, es eliminada. También presenta otro algoritmo, al que llama "5-window" que es una extensión del anterior, cogiendo conjuntos de 5 notas en lugar de conjuntos de 3, siempre que se pueda (en el caso de la segunda y penúltima nota, al tener solo una nota delante y detrás respectivamente, se cogen conjuntos de 4 notas). En todos estos algoritmos, la nota inicial y la final entran en la reducción. Otro método de reducción es el presentado por Gilbert y Conklin (2007), que utilizan una gramática libre de contexto ("context-free grammar") para reducir las melodías, pudiendo reducir éstas a distintos niveles.

En este trabajo, para hacer la reducción de las melodías, nos basaremos en la idea de las pendientes de Zhu y Kankanhalli (2002). Como hemos dicho antes, éstos representaban la melodía de forma gráfica, y trazaban líneas entre los máximos y mínimos locales, llamándolas pendientes, y utilizaban las pendientes para alinear las melodías. En nuestro caso, nos quedaremos con las notas que sean máximos y mínimos locales, y también con las notas repetidas. De esta manera, podríamos decir que eliminamos el camino y nos quedamos con el destino. Veamos como quedaría la melodía "Abiatu da bere bidean" al aplicarle el método de reducción. Para ello, mostraremos la melodía completa en primer lugar (sin aplicar la reducción, figura 2.1), y en segundo lugar mostraremos la misma melodía redondeando las notas que quedan al hacer la reducción (figura2.2).

ABIATU DA BERE BIDEAN

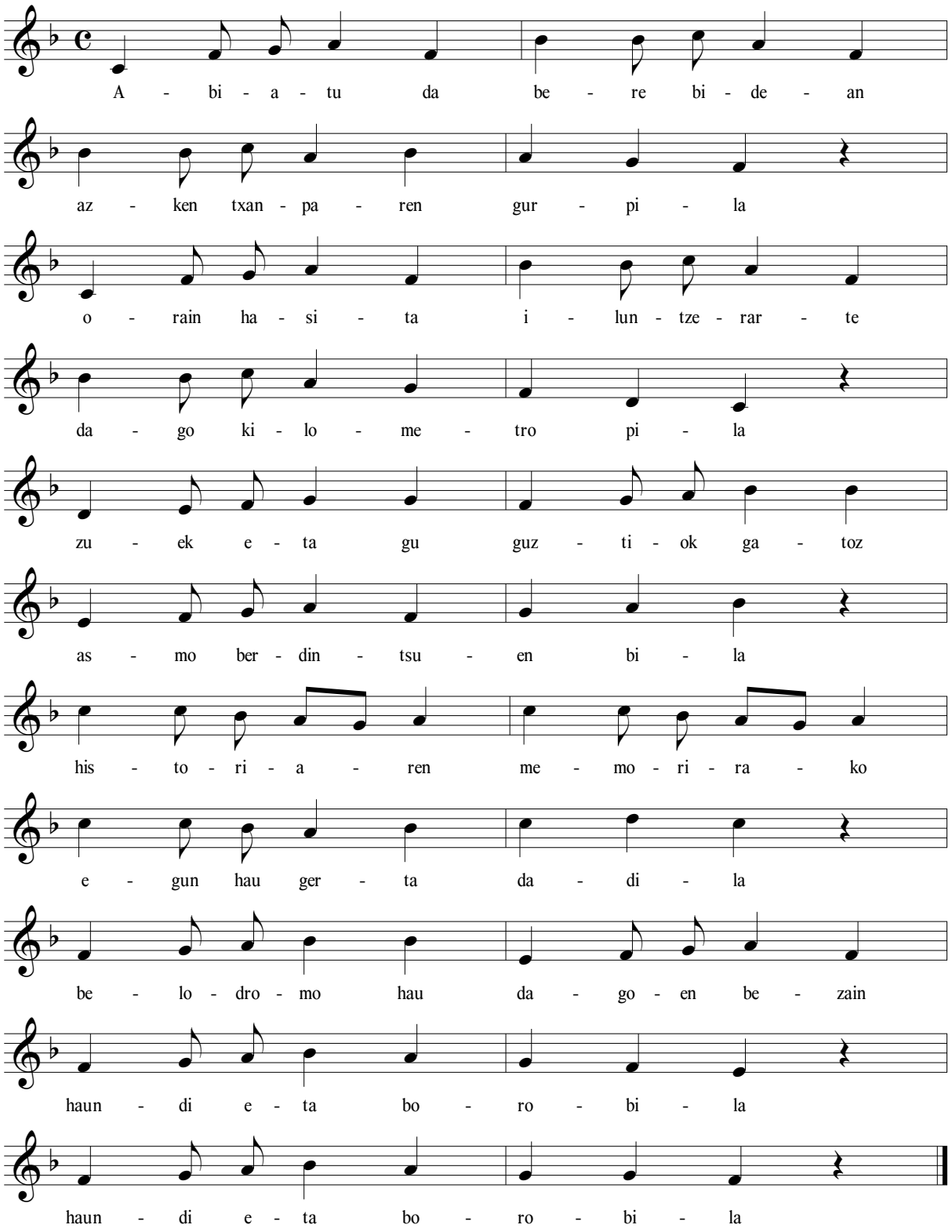


Figura 2.1: Partitura de la melodía “Abiatu da bere bidean” completa

ABIATU DA BERE BIDEAN

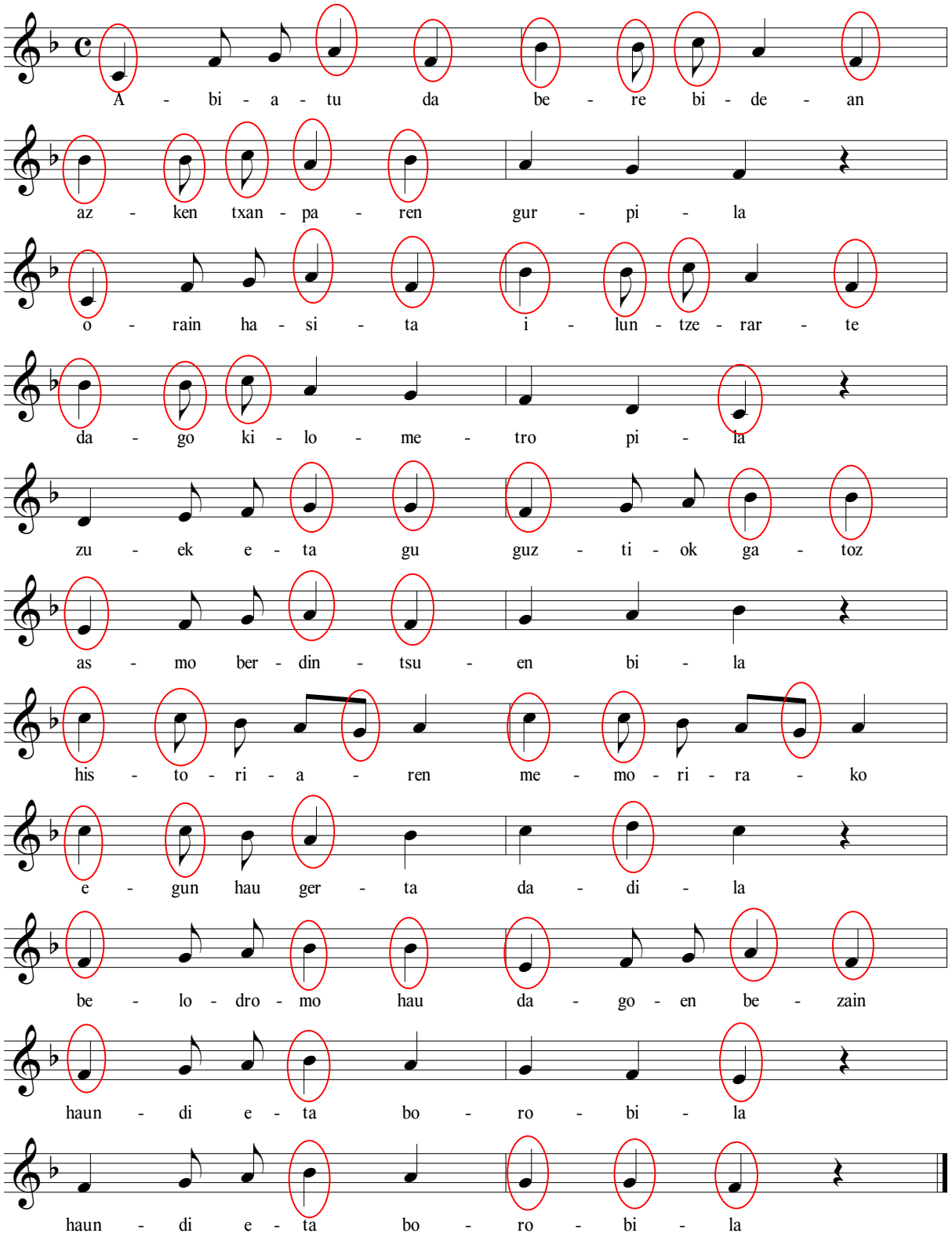


Figura 2.2: Partitura de la melodía “Abiatu da bere bidean” con las notas que quedan en la reducción resaltadas

Podemos apreciar que en esta partitura tenemos varios silencios, que como hemos dicho antes, no son faciles de identificar a la hora de leer los datos MIDI. En este caso, podemos decir que los silencios separan las frases de la melodía de forma clara, y podríamos pensar que las notas anteriores y posteriores a los silencios, que son final y comienzo de frase respectivamente, deberían entrar en la reducción. Sin embargo, puesto que no en todos los casos los silencios separan frases (puede que los silencios se introduzcan para ayudar a la expresividad), no podemos crear una regla general para que la reducción se quede con las notas anteriores y posteriores a los silencios aunque no sean máximos o mínimos locales.

2.5. Método Precision-Recall

El algoritmo Precision-Recall (o precisión y exhaustividad) es el método que emplearemos para comparar los resultados obtenidos por los algoritmos de alineamiento. El Precision-Recall es una métrica empleada en la medida del rendimiento de los sistemas de búsqueda y recuperación de información y reconocimiento de patrones. Se denomina precision a la fracción de instancias recuperadas que son relevantes, y recall a la fracción de instancias relevantes que han sido recuperadas.

Para realizar la comparación de resultados, aplicaremos este algoritmo sobre una variedad de threshold (o umbral). Ordenaremos los resultados de mejor a peor, quitando de cada fila el resultado de comparar una canción consigo misma, redimensionaremos los resultados para que queden entre 0 y 1, y calcularemos veinte puntos, comenzando por un threshold de $0,05$ y aumentándolo $0,05$ cada vez. El cálculo de los valores precision y recall se realizan de la siguiente manera:

$$precision(c,t) = \frac{(número\ de\ canciones\ de\ la\ familia\ a\ la\ que\ pertenece\ c) > t}{(número\ de\ canciones) > t}$$

$$recall(c,t) = \frac{(número\ de\ canciones\ de\ la\ familia\ a\ la\ que\ pertenece\ c) > t}{(número\ de\ canciones\ de\ la\ familia\ a\ la\ que\ pertenece\ c)}$$

donde c es la canción que tomamos como query y t es el valor de threshold.

Una vez calculados precision y recall para cada canción, se hace el promedio de todos los valores para cada valor de threshold, obteniendo dos vectores de 20 elementos que indican precision y recall para cada threshold. Es decir:

$$precision(t) = \frac{\sum_{\forall c} precision(c,t)}{(número\ de\ canciones)}$$

$$recall(t) = \frac{\sum_{\forall c} recall(c,t)}{(número\ de\ canciones)}$$

Una vez se tienen los vectores, se representan los resultados en un gráfico que tiene el siguiente aspecto:

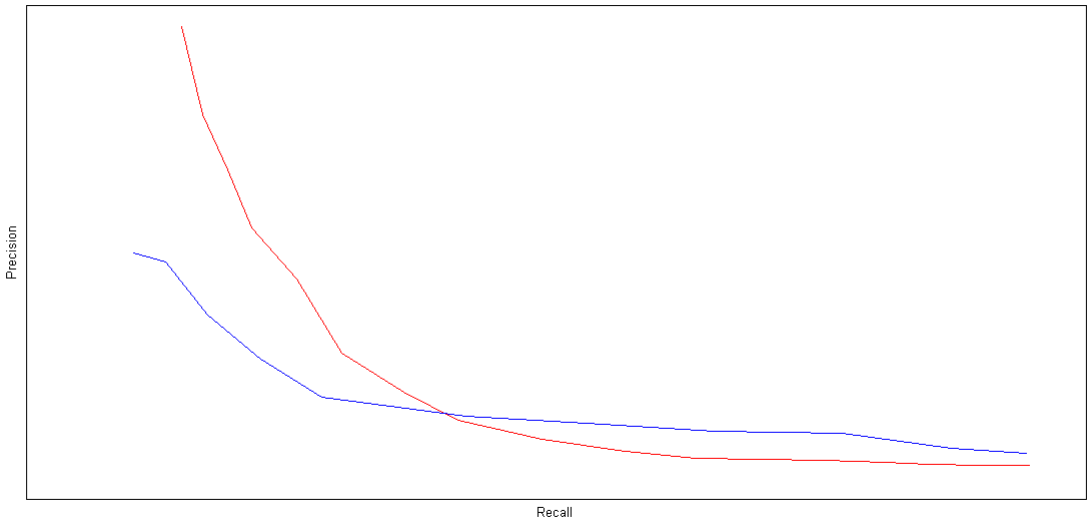


Figura 2.3: Ejemplo de gráfico de Precision-Recall que compara los resultados de dos representaciones de las melodías con el algoritmo Smith-Waterman (rojo:representación con notas, azul:representación con intervalos)

3

Resultados

En esta sección compararemos los resultados obtenidos con los métodos mencionados en el apartado anterior. Para cada comparación, explicaremos qué métodos o qué características de la melodía vamos a comparar, por qué hacemos dicha comparación (qué es lo que queremos ver), añadiremos su gráfico de Precision-Recall, y analizaremos brevemente los resultados obtenidos de cada comparación.

En primer lugar, vamos a comparar los métodos de alineamiento, es decir, el algoritmo de Smith-Waterman adaptado, el algoritmo de Needleman-Wunsch y el algoritmo Edit-Distance. Para ello, compararemos los resultados obtenidos con los tres métodos sobre cadenas de notas (figura 3.1) y cadenas de intervalos (figura 3.2). De esta manera, queremos ver cuál de los métodos nos da mejores resultados, y cuál de ellos es más adecuado para el trabajo que estamos realizando.

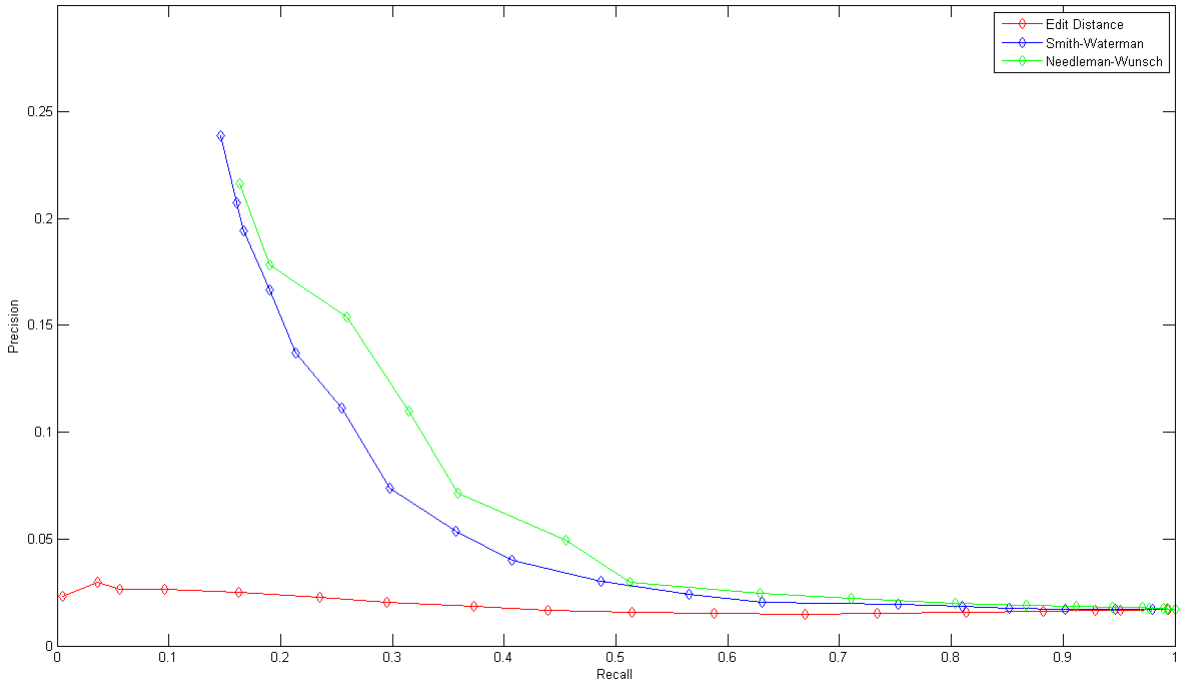


Figura 3.1: Curvas Precision-Recall de los algoritmos Smith-Waterman, Needleman-Wunsch y Edit-Distance sobre cadenas de notas

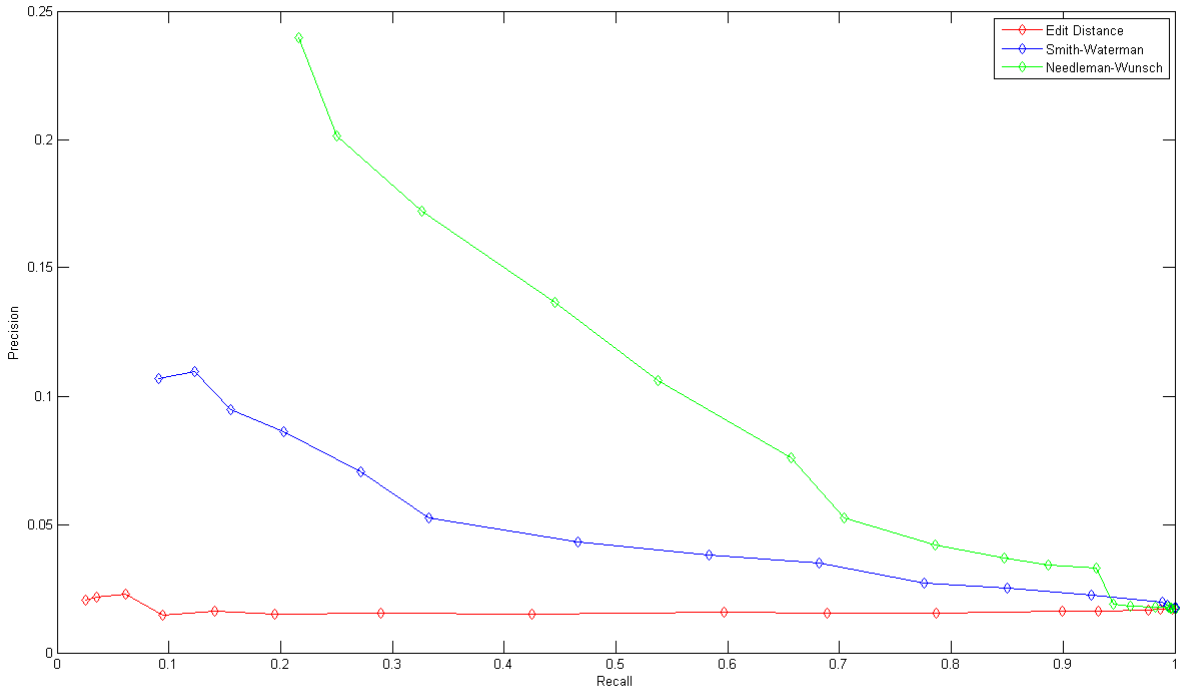


Figura 3.2: Curvas Precision-Recall de los algoritmos Smith-Waterman, Needleman-Wunsch y Edit-Distance sobre cadenas de intervalos

Podemos apreciar que en ambas figuras la curva Precision-Recall obtenida por el algoritmo Edit-Distance está por debajo de las demás, manteniendo un nivel bajo de Precision para todos

los valores de Recall. Además, en la figura 3.1 podemos ver que tanto el algoritmo de Smith-Waterman y el algoritmo de Needleman-Wunsch dan resultados similares, aunque los obtenidos por el algoritmo de Needleman-Wunsch son ligeramente mejores. Por otro lado, en la figura 3.2 podemos ver que la curva Precision-Recall de Needleman-Wunsch está muy por encima de la curva obtenida por el algoritmo Smith-Waterman. Volveremos a comparar estos algoritmos al final de esta sección.

A continuación compararemos los resultados de las melodías reducidas y sin reducir. Con esta comparación veremos si lo más importante es el destino de las notas (es decir, la nota máxima o mínima local) o tiene importancia el camino hasta llegar a esas notas (las notas que hay entre esos máximos y mínimos locales). Para ver esto, igual que en el caso anterior, compararemos los resultados obtenidos en cadenas de notas y cadenas de intervalos de las melodías con y sin reducción, utilizando el algoritmo de Smith-Waterman.

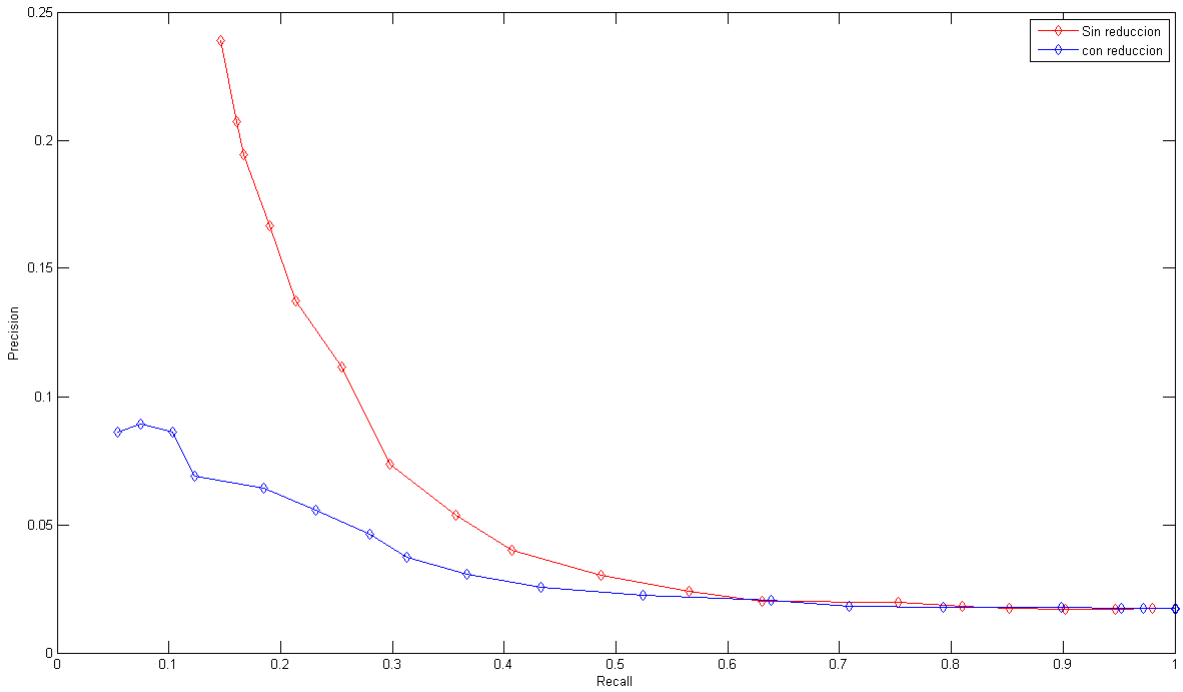


Figura 3.3: Curvas Precision-Recall del algoritmo Smith-Waterman sobre cadenas de notas con y sin reducción

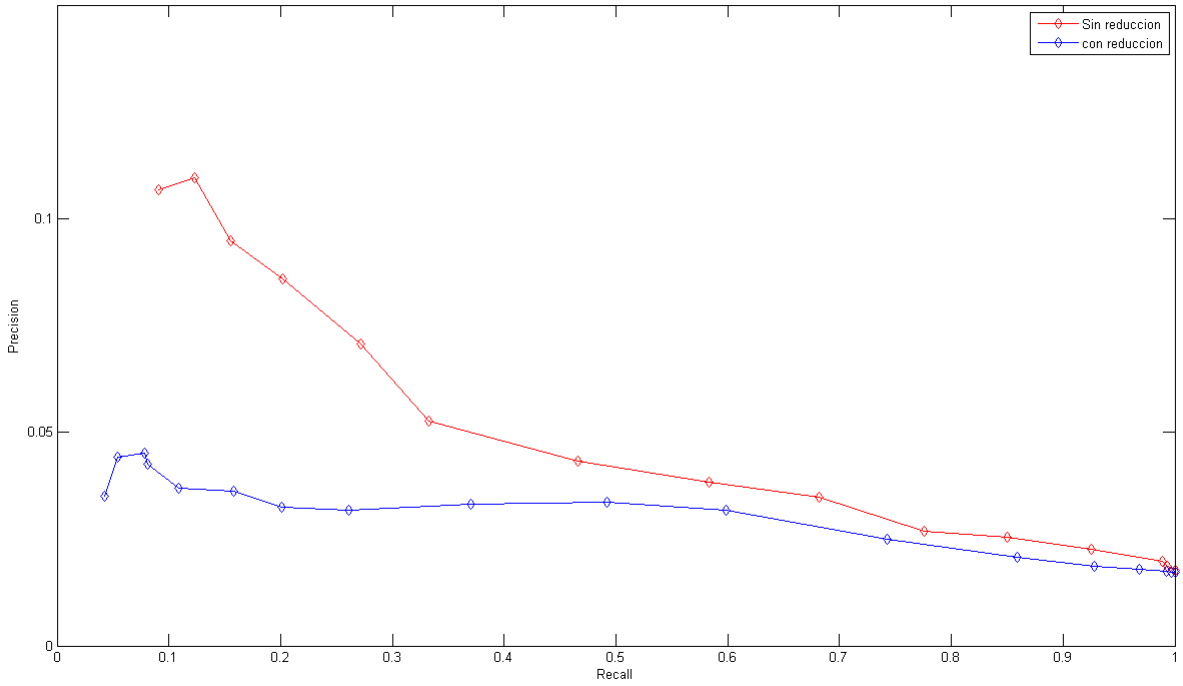


Figura 3.4: Curvas Precision-Recall del algoritmo Smith-Waterman sobre cadenas de intervalos con y sin reducción

Tanto en la figura 3.3 como en la figura 3.4 podemos ver que las curvas obtenidas por las cadenas sin reducción están por encima de las curvas obtenidas con las cadenas sacadas de las melodías reducidas. Esto nos indica que los resultados obtenidos con las melodías sin reducir son mejores que los resultados obtenidos con las melodías reducidas. Esto puede significar que la reducción que estamos utilizando elimina información relevante de las melodías.

Ya hemos visto que los algoritmos de Smith-Waterman y Needleman-Wunsch son más adecuados que el algoritmo Edit-Distance para calcular la similitud entre melodías y que las melodías completas dan mejores resultados que las melodías reducidas. Ahora veamos qué tipo de representación es más apropiada para realizar el cálculo de similitudes. Para ello compararemos los resultados obtenidos por la representación en cadena de notas, la representación en cadena de intervalos y la representación en cadena de notas teniendo en cuenta la duración, utilizando los algoritmos de Smith-Waterman y Needleman-Wunsch, representando sus respectivas curvas Precision-Recall de dos en dos en primer lugar, y después representaremos las tres curvas en el mismo gráfico para verlo más claro. Para finalizar, representaremos las curvas las distintas representaciones y distintos algoritmos en un mismo gráfico.

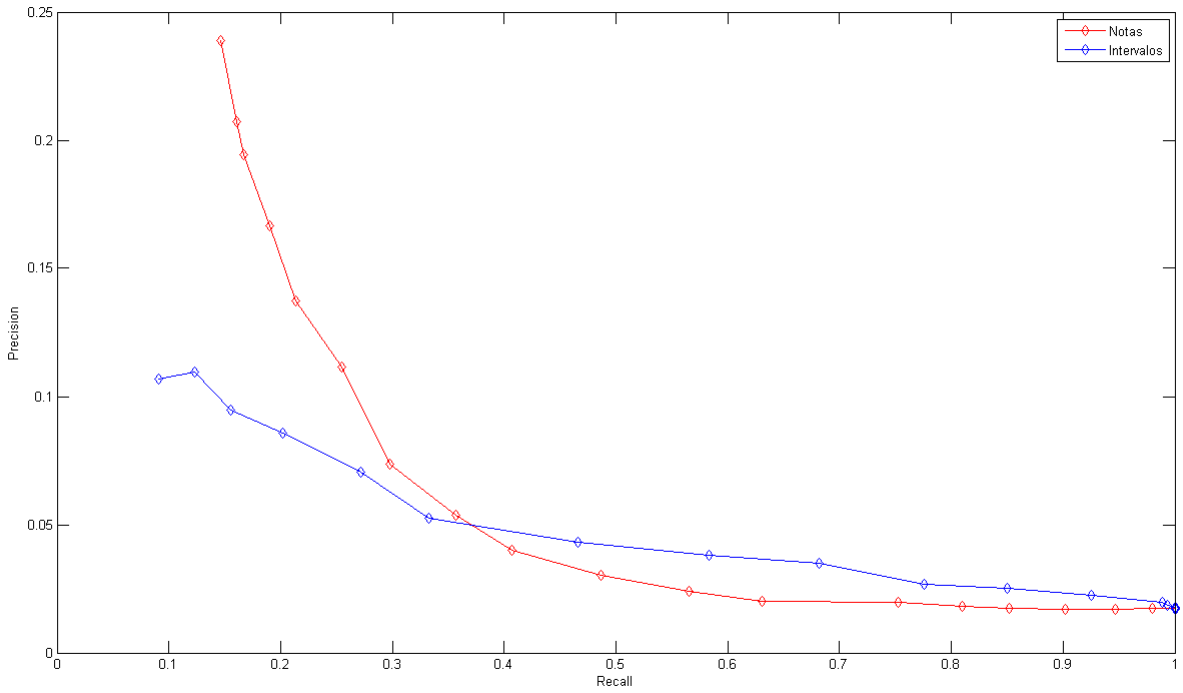


Figura 3.5: Curvas Precision-Recall del algoritmo Smith-Waterman sobre cadenas de notas y cadenas de intervalos

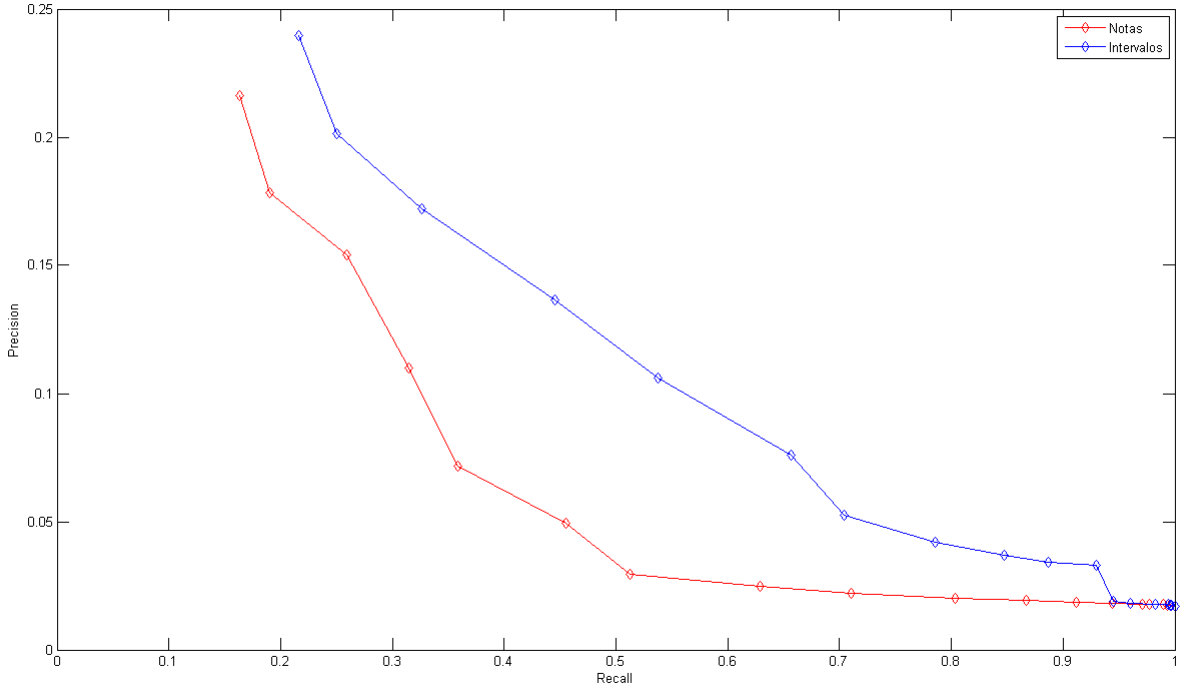


Figura 3.6: Curvas Precision-Recall del algoritmo Needleman-Wunsch sobre cadenas de notas y cadenas de intervalos

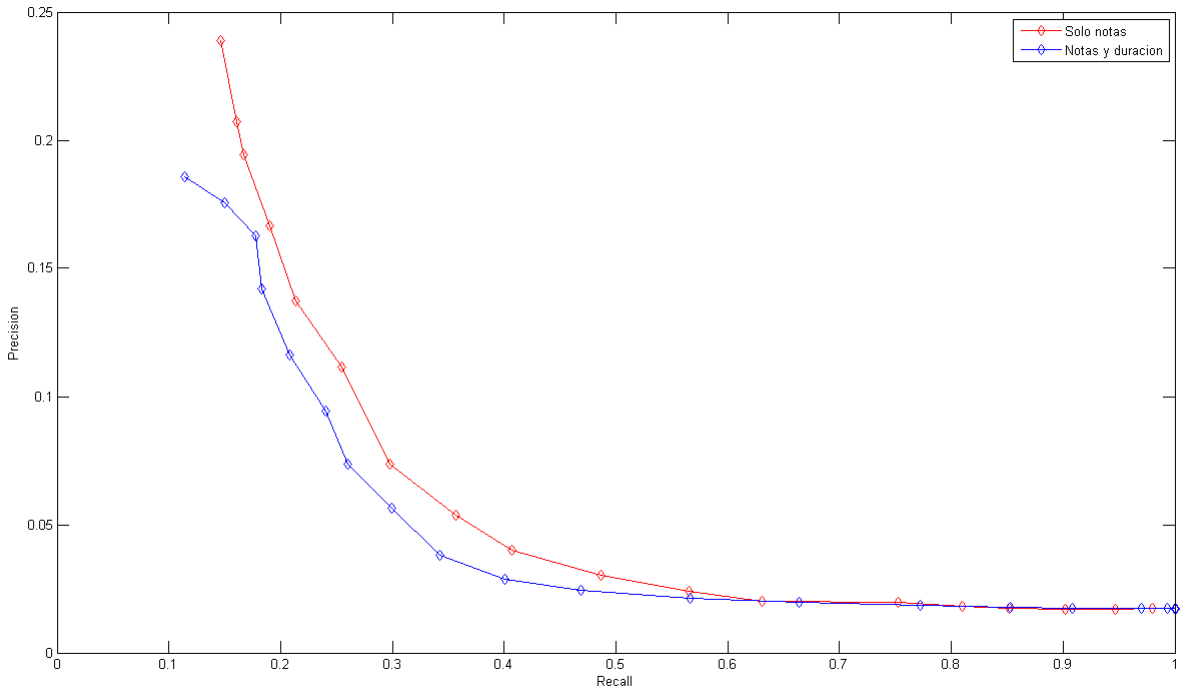


Figura 3.7: Curvas Precision-Recall del algoritmo Smith-Waterman sobre cadenas de notas y cadenas de notas + duración

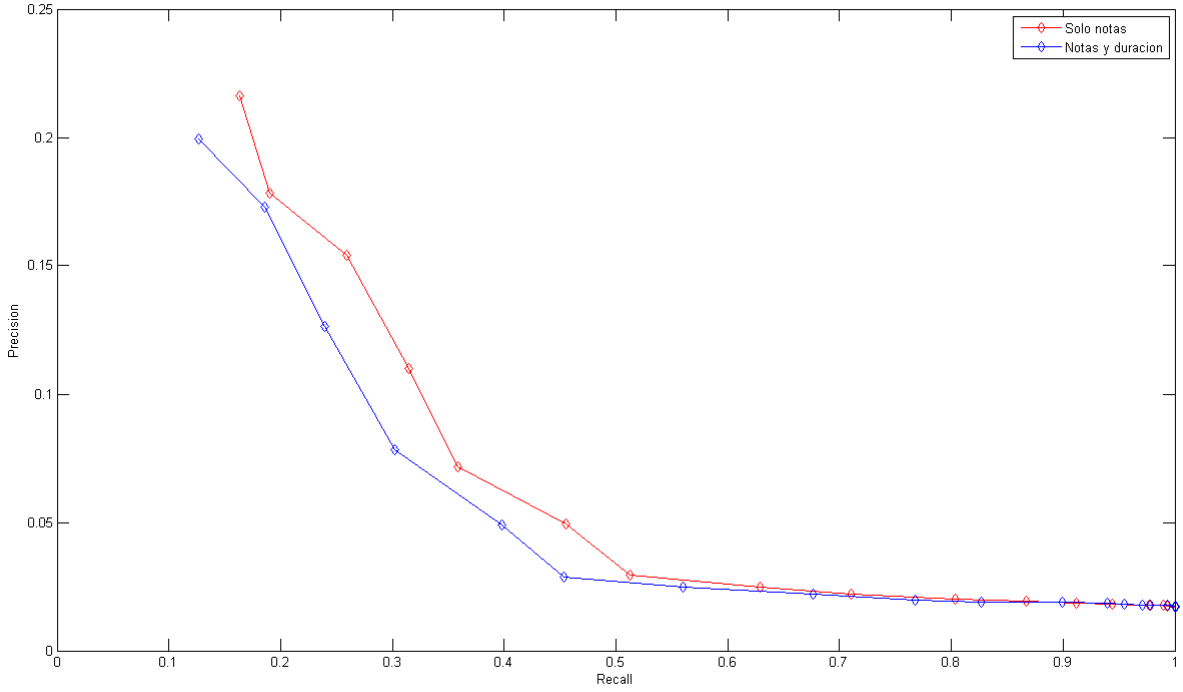


Figura 3.8: Curvas Precision-Recall del algoritmo Needleman-Wunsch sobre cadenas de notas y cadenas de notas + duración

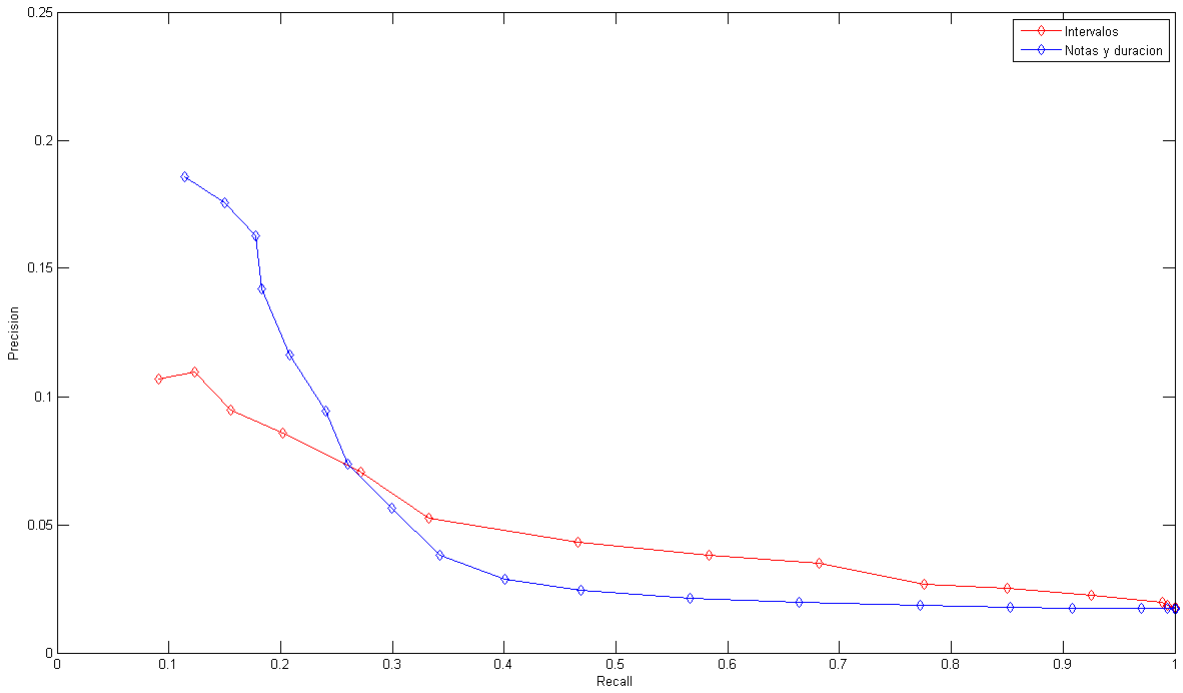


Figura 3.9: Curvas Precision-Recall del algoritmo Smith-Waterman sobre cadenas de notas + duración y cadenas de intervalos

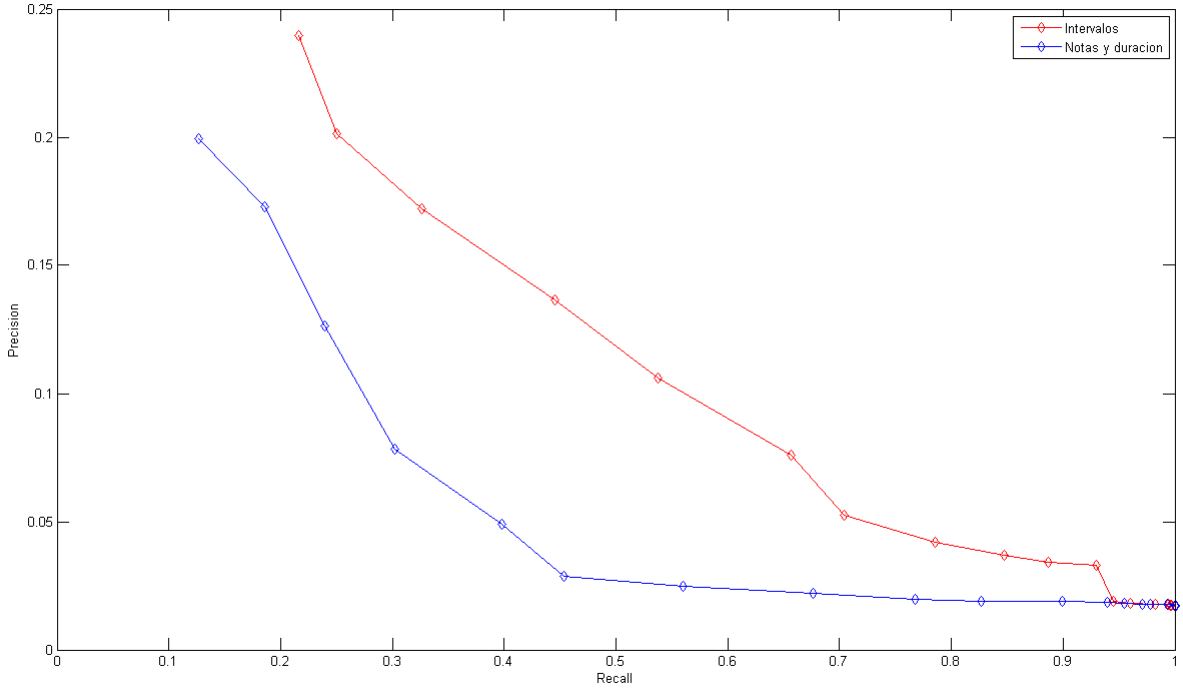


Figura 3.10: Curvas Precision-Recall del algoritmo Needleman-Wunsch sobre cadenas de notas + duración y cadenas de intervalos

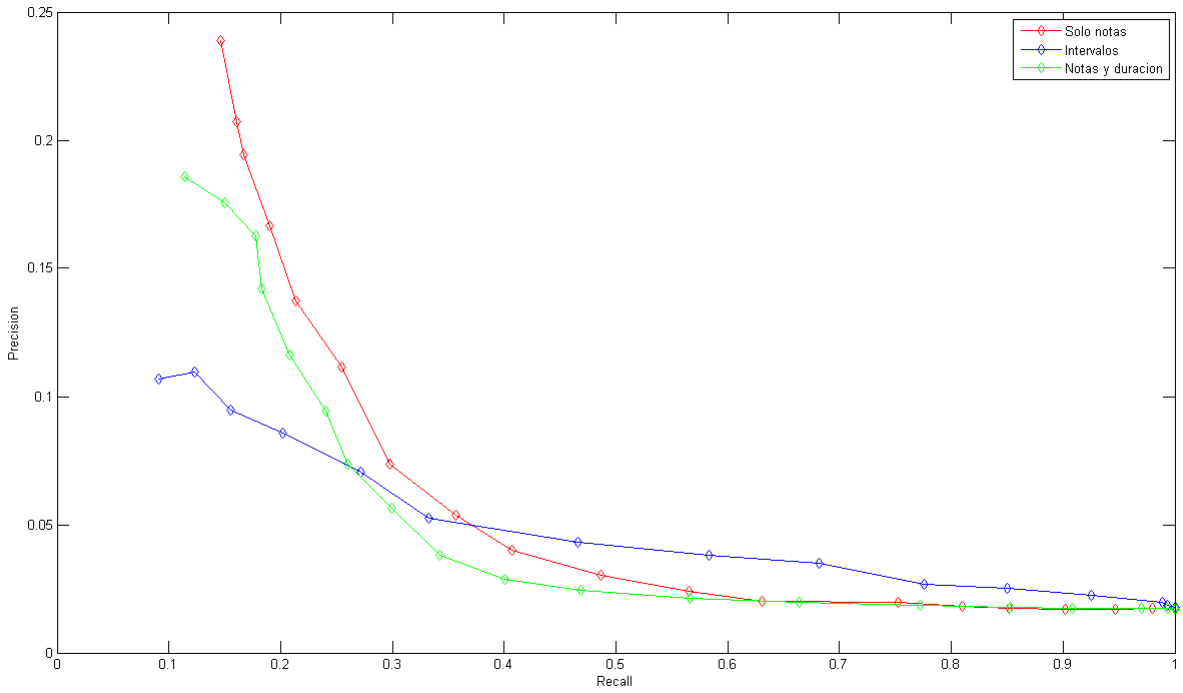


Figura 3.11: Curvas Precision-Recall del algoritmo Smith-Waterman sobre cadenas de notas, cadenas de notas + duración y cadenas de intervalos

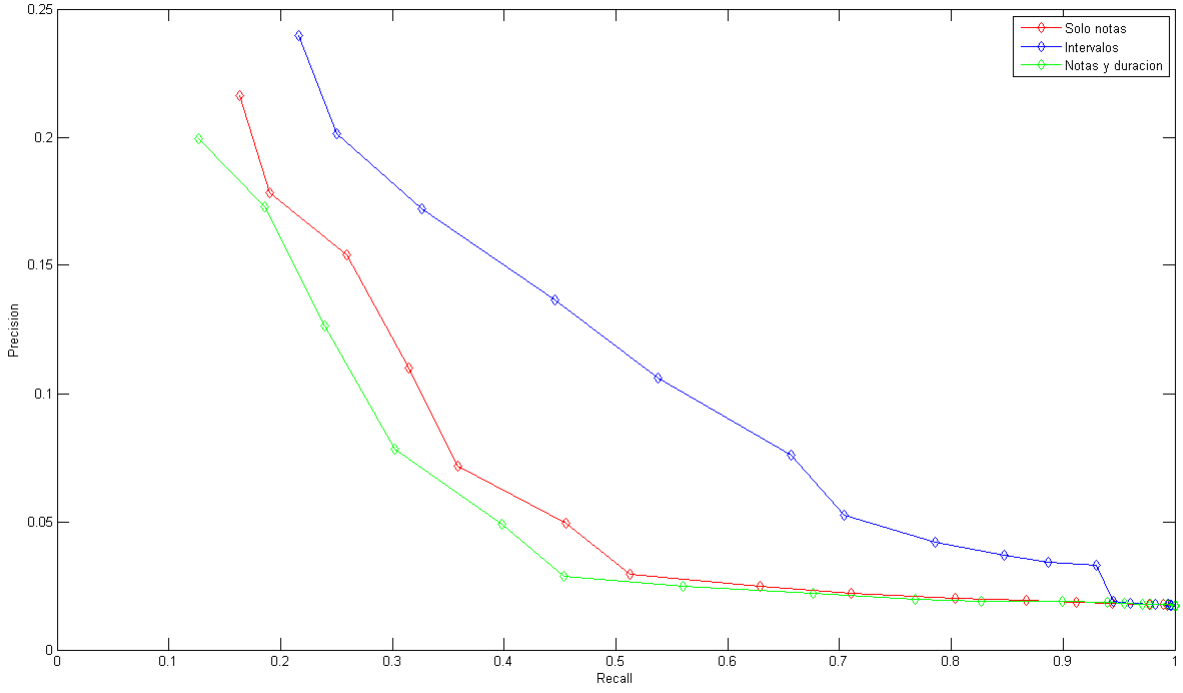


Figura 3.12: Curvas Precision-Recall del algoritmo Needleman-Wunsch sobre cadenas de notas, cadenas de notas + duración y cadenas de intervalos

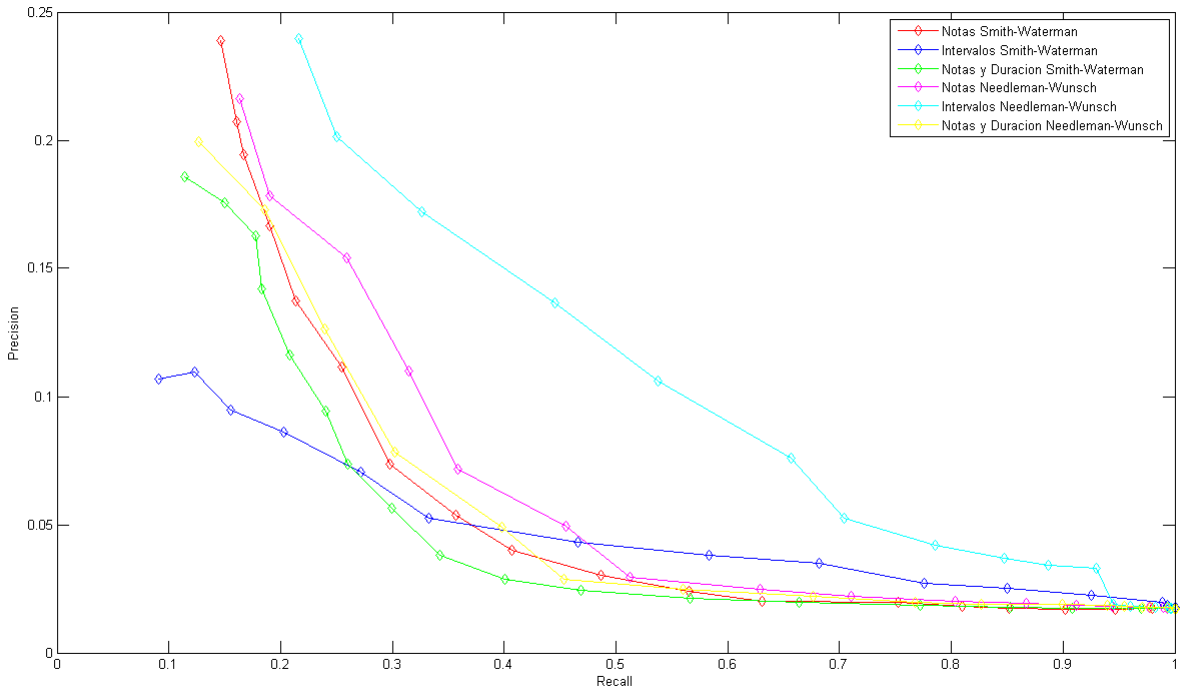


Figura 3.13: Curvas Precision-Recall de los algoritmos Smith-Waterman y Needleman-Wunsch sobre cadenas de notas y cadenas de intervalos

En el caso de comparar las representaciones con cadenas de notas y cadenas de notas con duración (figuras 3.7 y 3.8) vemos que los resultados obtenidos por la primera son mejores, ya que aunque al final ambas curvas van prácticamente a la par, en valores de Recall menores los valores de Precision son mejores. En los casos en los que comparamos cadenas de notas, ya sea con duración (figuras 3.9 y 3.10) o sin duración (figuras 3.5 y 3.6) con cadenas de intervalos, vemos que en el caso del algoritmo Smith-Waterman, para valores bajos de Recall, la Precision es mayor en las representaciones con cadenas de notas. Sin embargo, para valores no muy altos de Recall (algo menor $0,4$ en el caso en el que comparamos cadenas intervalos con cadenas de notas sin duración, y algo menor que $0,3$ en el caso en el que comparamos cadenas de intervalos con cadenas de notas con duración), vemos que la curva de la cadena de intervalos pasa a estar por encima en ambos casos. En el caso del algoritmo de Needleman-Wunsch, la curva Precision-Recall obtenida con la representación de intervalos está muy por encima de las otras curvas. Debido a esto, podemos decir que la representación más adecuada para calcular la similitud entre melodías es la de cadenas de intervalos.

Por otro lado, en el gráfico 3.13, vemos que para las tres distintas representaciones las curvas del algoritmo Needleman-Wunsch están por encima de la curva correspondiente a la misma representación con el algoritmo de Smith-Waterman.

4

Conclusiones

En el apartado anterior hemos visto los resultados que se obtienen utilizando distintos métodos, representaciones y características de la melodía, y hemos mencionado brevemente lo que se puede ver. En esta sección explicaremos con más profundidad lo obtenido antes.

En primer lugar, hemos visto que los algoritmos Smith-Waterman y Needleman-Wunsch dan mejores resultados que el algoritmo Edit-Distance. Como ya hemos mencionado, los dos primeros algoritmos miden la similitud entre melodías mientras que Edit-Distance, como el mismo nombre indica, mide la distancia entre éstas, por lo que es necesario hacer una normalización de estas medidas para poder comparar los resultados de ambos algoritmos. Por eso, es importante hacer una buena normalización de ambas medidas para que la comparación de las dos sea correcta. Por otra parte, el algoritmo Edit-Distance asigna dos valores: 0 en caso de que los caracteres coincidan y 1 en el resto de situaciones, mientras que el algoritmo Smith-Waterman y el algoritmo de Needleman-Wunsch dan una puntuación máxima en caso de que los caracteres coincidan, pero la penalización en caso de que no sea así varía dependiendo de los caracteres. Debido a esto, los resultados de estos algoritmos son más concretos. Por ejemplo, si tenemos tres melodías que difieren en una sola nota, donde en una esa nota es C, en otra es D y en otra es F, la distancia entre las tres melodías cogidas de dos en dos medida por Edit-Distance es igual, mientras que los resultados de los algoritmo Smith-Waterman y Needleman-Wunsch son distintas en las tres comparaciones.

Además, hemos apreciado que el algoritmo de Needleman-Wunsch ha dado mejores resultados que el algoritmo de Smith-Waterman, que da resultados ligeramente mejores en el caso de las representaciones con cadenas de notas, con y sin duración, y bastante mejores en el caso de la representación por cadenas de intervalos. Sin embargo, como ya hemos mencionado, el algoritmo de Needleman-Wunsch alinéa las secuencias de manera global, mientras que el algoritmo de Smith-Waterman las alinéa de manera local. Como en la colección de melodías que tenemos hay melodías de longitudes muy distintas, el algoritmo de Smith-Waterman es más adecuado que el de Needleman-Wunsch, ya que el algoritmo de Needleman-Wunsch penaliza la diferencia de longitud entre las melodías.

En segundo lugar, los resultados nos indican que las melodías sin reducir dan mejores resultados que las melodías reducidas. Esto nos indica que la reducción que hemos implementado elimina información relevante de la melodía para comparar las familias. Además, la dificultad de poder crear una regla que nos indique si los silencios son final de frase o son silencios expresivos

nos imposibilita añadir notas que sean final o inicio de frase sin que sean máximos o mínimos locales o notas repetidas. Por lo tanto, al hacer la reducción eliminamos notas que pueden ser importantes en la melodía.

Para finalizar, hemos visto que la representación por intervalos da mejores resultados que las representaciones por notas, teniendo en cuenta la duración y sin tener en cuenta la duración. Como hemos mencionado en la introducción, hay varios requisitos que deben cumplir los métodos de comparación de melodías (Urbano et al., 2011). La representación de las melodías por intervalos es invariante de transposición, es decir, cumple los siguientes tres requisitos: aceptar como iguales melodías que solo difieran en la octava en la que han sido escritas, o que estando escritas en distintas tonalidades tengan las mismas notas, o que cambiando de tonalidad, las notas sean del mismo grado en su tonalidad respectiva. Por su parte, las representaciónes de las melodías utilizando notas que hemos aplicado requieren cambiar la tonalidad de las melodías a Do Mayor, para que todas tengan la misma tonalidad. Sin embargo, puede que además de la tonalidad haya notas con alteraciones (sostenidos o bemoles) que afecten a la melodía. Se dan casos en los que sin que una alteración este reflejada en la armadura (alteraciones predefinidas que dependen de la tonalidad) durante toda la melodía una nota esté escrita con dicha alteración, por lo que, en la mayoría de los casos, podríamos pasar esa alteración a la armadura, cambiando la tonalidad original. Esto hace que al pasar una melodía en la que ocurra esto a Do Mayor, siga habiendo una nota alterada en toda la melodía, por lo que en realidad la tonalidad es otra. Además, si tenemos dos partituras que tienen las mismas notas en sus respectivas tonalidades (siendo distintas), al cambiar sus tonalidades a Do Mayor son melodías con notas totalmente distintas, por lo que no cumple el segundo de los requisitos mencionados.

A su vez, la representación por notas sin duración da mejores resultados que la representación por notas teniendo en cuenta la duración. Esto puede ser debido a que al no tener en cuenta la duración, lo que tenemos es una cadena que indica las notas de la melodía y su orden, y el no añadir silencios no afecta mucho en esto. Sin embargo, al tener en cuenta la duración de las notas, los silencios ganan importancia, y al no añadirlos, la duración de la melodía se ve afectada, pudiende hacer que dos melodías que originariamente tenían la misma duración no la tengan con esta representación.

Bibliografía

Aloupis, G., Fevens, T., Langerman, S., Matsui, T., Mesa, A., Rodríguez, Y. N., Rappaport, D., and Toussaint, G. T. (2006). Algorithms for computing geometric measures of melodic similarity. *Computer Music Journal*, 30(3):67–76.

Bor, M. (2009). *Contour Reduction Algorithms: a Theory of Pitch and Duration Hierarchies for Post-Tonal Music*. PhD thesis, University of British Columbia (Vancouver).

De Bortoli, A. (2009). Music Information Retrieval (MIR). Approaches for content-based music retrieval. *http://es.slideshare.net/albertodebortoli/music-information-retrieval*.

Flexer, A., Gouyon, F., Dixon, S., and Widmer, G. (2006). Probabilistic combination of features for music classification. In *ISMIR 2006, 7th International Conference on Music Information Retrieval, Victoria, Canada, 8-12 October 2006, Proceedings*, pages 111–114.

Frey, J. D. (2008). Finding song melody similarities using a DNA string matching algorithm. Master's thesis, Kent State University.

Garzia, J., Sarasua, J., and Egaña, A. (2001). *The Art of Bertsolaritza: Improvised Basque Verse Singing*. Bertsolari Liburuak.

Gilbert, E. and Conklin, D. (2007). A Probabilistic Context-Free Grammar for Melodic Reduction. In *Proceedings of the International Workshop on Artificial Intelligence and Music, 20th International Joint Conference on Artificial Intelligence (IJCAI), Hyderabad, India.*, pages 83–94.

Gouyon, F. and Dixon, S. (2005). Influence of input features in perceptual tempo induction. In *2nd Annual Music Information Retrieval eXchange (MIREX)*.

Grachten, M., Arcos, J. L., and de Mántaras, R. L. (2004). Melodic similarity: Looking for a good abstraction level. In *ISMIR 2004, 5th International Conference on Music Information Retrieval, Barcelona, Spain, October 10-14, 2004, Proceedings*, pages 210–215.

Hu, N., Dannenberg, R. B., and Lewis, A. L. (2002). A probabilistic model of melodic similarity. In *Proceedings of the International Computer Music Conference*, pages 471–474. International Computer Music Society.

Janssen, B., van Kranenburg, P., and Volk, A. (2015). A comparison of symbolic similarity measures for finding occurrences of melodic segments. In *Proceedings of the 16th International Society for Music Information Retrieval Conference, ISMIR 2015, Málaga, Spain, October 26-30, 2015*, pages 659–665.

Mongeau, M. and Sankoff, D. (1990). Comparison of musical sequences. *Computers and the Humanities*, 24(3):161–175.

Morris, R. D. (1993). New directions in the theory and analysis of musical contour. *Music Theory Spectrum*, 15(2):205–228.

Müllensiefen, D. and Frieler, K. (2004). Optimizing measures of melodic similarity for the exploration of a large folk song database. In *ISMIR 2004, 5th International Conference on Music Information Retrieval, Barcelona, Spain, October 10-14, 2004, Proceedings*, pages 274–280.

Narmour, E. (1990). The analysis and cognition of basic melodic structures : the implication-realization model. *University of Chicago Press*.

Narmour, E. (1992). The analysis and cognition of melodic complexity: the implication-realization model. *University of Chicago Press*.

Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453.

Ó Maidín, D. S. (1998). A geometrical algorithm for melodic difference. *Computing in Musicology*, 11:65–72.

Orio, N. and Rodà, A. (2009). A measure of melodic similarity based on a graph representation of the music structure. In *Proceedings of the 10th International Society for Music Information Retrieval Conference, ISMIR 2009, Kobe International Conference Center, Kobe, Japan, October 26-30, 2009*, pages 543–548.

Savage, P. E. and Atkinson, Q. D. (2015). Automatic tune family identification by musical sequence alignment. In *Proceedings of the 16th International Society for Music Information Retrieval Conference, ISMIR 2015, Málaga, Spain, October 26-30, 2015*, pages 162–168.

Smith, T. F. and Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197.

Typke, R., Giannopoulos, P., Veltkamp, R. C., Wiering, F., and van Oostrum, R. (2003). Using transportation distances for measuring melodic similarity. In *ISMIR 2003, 4th International Conference on Music Information Retrieval, Baltimore, Maryland, USA, October 27-30, 2003, Proceedings*, pages 107–114.

Urbano, J. (2014). MelodyShape at MIREX 2014 Symbolic Melodic Similarity. Technical report, Music Information Retrieval Evaluation eXchange.

Urbano, J., Lloréns, J., Morato, J., and Sánchez-Cuadrado, S. (2011). Melodic Similarity through Shape Similarity. In Ystad, S., Aramaki, M., Kronland-Martinet, R., and Jensen, K., editors, *Exploring Music Contents*, pages 338–355. Springer.

van Kranenburg, P., Volk, A., and Wiering, F. (2013). A comparison between global and local features for computational classification of folk song melodies. *Journal of New Music Research*, 42(1):1–18.

Wallentinsen, K. M. (2013). A Hierarchical Approach to the Analysis of Intermediary Structures Within the Modified Contour Reduction Algorithm. Master's thesis, University of Massachusetts Amherst.

Zhu, Y., Kankanhalli, M. S., and Tian, Q. (2002). Similarity matching of continuous melody contours for humming querying of melody databases. In *IEEE 5th Workshop on Multimedia Signal Processing, MMSP 2002, St. Thomas, Virgin Islands, USA, December 9-11, 2002*, pages 249–252.

Zhu, Y., Xu, C., and Kankanhalli, M. S. (2001a). Melody curve processing for music retrieval. In *Proceedings of the 2001 IEEE International Conference on Multimedia and Expo, ICME 2001, August 22-25, 2001, Tokyo, Japan*, pages 285–288.

Zhu, Y., Xu, C., and Kankanhalli, M. S. (2001b). Pitch tracking and melody slope matching for song retrieval. In *Advances in Multimedia Information Processing - PCM 2001, Second IEEE Pacific Rim Conference on Multimedia, Bejing, China, October 24-26, 2001, Proceedings*, pages 530–537.

Apéndice A

Lectura de archivos MIDI y guardar output en archivo *.txt (Java

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;

import javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.MidiEvent;
import javax.sound.midi.MidiMessage;
import javax.sound.midi.MidiSystem;
import javax.sound.midi.Sequence;
import javax.sound.midi.ShortMessage;
import javax.sound.midi.Track;


public class Main {

        private ArrayList<Integer> notes;

    public static final int NOTE_ON = 0x90;
    public static final int NOTE_OFF = 0x80;
    public static final String[] NOTE_NAMES = {"C", "C#", "D", "D#", "E", "F", "
        F#", "G", "G#", "A", "A#", "B"};

        public static void main(String[] args) throws Exception {


                Sequence sequence = MidiSystem.getSequence(new File(
                                "/Users/Sarriegi/Desktop/melodik/liburuki5_2688a
                                    .mid"));

                ArrayList<Integer> notes = new ArrayList<Integer>(); // we don't
                    know
```

```
ArrayList<Integer> duration = new ArrayList<Integer>();

FileWriter fw = new FileWriter("/Users/Sarriegi/Desktop/Abiatu.
    txt",true);

int durationon = 0;
int durationoff = 0;

int noteNumber = 0;
int trackNumber = 0;
for (Track track : sequence.getTracks()) {
        trackNumber++;
        System.out.println("Track " + trackNumber + ": size =" 
                        + track.size());
        System.out.println();

        for (int i=0; i < track.size(); i++) {
MidiEvent event = track.get(i);
System.out.print("@" + event.getTick() + " ");
//fw.write("@" + event.getTick() + " ");
MidiMessage message = event.getMessage();
if (message instanceof ShortMessage) {
    ShortMessage sm = (ShortMessage) message;
    System.out.print("Channel: " + sm.getChannel() + " ");
    //fw.write("Channel: " + sm.getChannel() + " ");
    if (sm.getCommand() == NOTE_ON) {
        int key = sm.getData1();
        int octave = (key / 12)-1;
        int note = key % 12;
        String noteName = NOTE_NAMES[note];
        int velocity = sm.getData2();
        System.out.println("Note on, " + noteName + octave + " 
            key=" + key + " velocity: " + velocity);
        fw.write("@" + event.getTick() + " " + "Channel: " + sm.
            getChannel() + " " + "Note on, " + noteName + octave
            + " key=" + key + " velocity: " + velocity + "\r\n");
    }
    if (sm.getCommand() == NOTE_OFF) {
        int key = sm.getData1();
        int octave = (key / 12)-1;
        int note = key % 12;
        String noteName = NOTE_NAMES[note];
        int velocity = sm.getData2();
        System.out.println("Note off, " + noteName + octave + " 
            key=" + key + " velocity: " + velocity);
        fw.write("@" + event.getTick() + " " + "Channel: " + sm.
            getChannel() + " " + "Note off, " + noteName + octave
             + " key=" + key + " velocity: " + velocity + "\r\n")
            ;
        notes.add(key);
```

```
                    noteNumber++;
                }
            } else {
                //System.out.println("Other message: " + message.getClass()
                    ;
                //fw.write("Other message: " + message.getClass() + "\r\n");
            }

                    }
                    //for (int i = 0; i < noteNumber; i++) {
                            //System.out.println(notes.get(i));
                    //}
            //fwED.write(cod + "\r\n");
            //fwED.close();

            }
            fw.close();

        }

}
```

Apéndice B

Extracción de cadenas de caracteres (Matlab)

B.1. Representación por notas

```
function [CadenaNotas]=ObtenerCadenaNotas(ficherotexto,tonalidad,valornegra)
%Leemos el archivo
archivo=textread(ficherotexto,'%s','delimiter','\n');
%Creamos una matriz para quedarnos con la informacion relevante de cada
%linea de texto
M=zeros(length(archivo));
%Guardamos la informacion relevante en la matriz
for i=1:length(archivo)
    s=archivo{i};
    %Guardamos la nota
    for k=1:length(s)
        if s(k)=='='
            t=k;
        end
    end
    M(i)=str2num(s((t+1):(t+3)));
end
%Definimos un vector con los valores de tick de duracion y tonalidad para
%despues definir los vectores para construir las cadenas de notas
%c=[Valor tick negra, tonalidad]
c=[valornegra tonalidad];
%La dimension de la matriz que necesitamos para construir las cadenas de
%notas es de la mitad de filas que la matriz anterior y dos columnas
h=length(M);
l=h/2;
N=zeros(l,1);
%Hacemos la trasportacion de las notas
for i=1:l
    N(i)=M(2*i)+c(2);
end
%Una vez tenemos la matriz definida, solo queda crear la cadena de notas
CadenaNotas='';
for i=1:l
    %Definimos el caracter de cada nota
    vn=mod(N(i),12); %vn=valornota
    if vn==0
        cn='C'; %cn=caracternota
    elseif vn==1
```

```
        cn='V';
    elseif vn==2
        cn='D';
    elseif vn==3
        cn='W';
    elseif vn==4
        cn='E';
    elseif vn==5
        cn='F';
    elseif vn==6
        cn='X';
    elseif vn==7
        cn='G';
    elseif vn==8
        cn='Y';
    elseif vn==9
        cn='A';
    elseif vn==10
        cn='Z';
    elseif vn==11
        cn='B';
    end
    %Creamos la cadena de caracteres
    CadenaNotas=strcat(CadenaNotas,cn);
end
end
```

B.2. Representación por notas con reducción

```
function[CadenaNotas]=ObtenerCadenaNotasReduccion(ficherotexto,tonalidad,
    valornegra)
%Leemos el archivo
archivo=textread(ficherotexto,'%s','delimiter','\n');
%Creamos una matriz para quedarnos con la informacion relevante de cada
%linea de texto
M=zeros(length(archivo));
%Guardamos la informacion relevante en la matriz
for i=1:length(archivo)
    s=archivo{i};
    %Guardamos la nota
    for k=1:length(s)
        if s(k)=='='
            t=k;
        end
    end
    M(i)=str2num(s((t+1):(t+3)));
end
%Definimos un vector con los valores de tick de duracion y tonalidad para
%despues definir los vectores para construir las cadenas de notas
%c=[Valor tick negra, tonalidad]
c=[valornegra tonalidad];
%La dimension de la matriz que necesitamos para construir las cadenas de
%notas es de la mitad de filas que la matriz anterior
l=h/2;
N=zeros(l,1);
%Hacemos la trasportacion de las notas
for i=1:l
    N(i)=M(2*i)+c(2);
end
%Hacemos la reduccion
```

```
%Definimos el contorno de la melodia
contorno=zeros(l,1);
for i=2:l
    if(N(i)-N(i-1))>0
        contorno(i)=1;
    elseif(N(i)-N(i-1))<0
        contorno(i)=-1;
    elseif(N(i)-N(i-1))==0
        contorno(i)=0;
    end
end
%Nos quedamos con las notas repetidas, maximos y minimos locales, la
%primera y la ultima
reduccion=[1];
nnr=1; %numero de notas que entran en la reduccion
for i=2:(l-1)
    if contorno(i)==0
        reduccion=[reduccion i];
        nnr=nnr+1;
    elseif (contorno(i)*contorno(i+1))<=0
        reduccion=[reduccion i];
        nnr=nnr+1;
    end
end
reduccion=[reduccion l];
nnr=nnr+1;
%Creamos un vector con las notas de la reduccion
N1=zeros(nnr,1);
for i=1:nnr
    aux=reduccion(i);
    N1(i)=N(aux);
end
%Una vez tenemos la matriz definida, solo queda crear la cadena de notas
CadenaNotas=' ';
for i=1:nnr
    %Definimos el caracter de cada nota
    vn=mod(N1(i),12); %vn=valornota
    if vn==0
        cn='C'; %cn=caracternota
    elseif vn==1
        cn='V';
    elseif vn==2
        cn='D';
    elseif vn==3
        cn='W';
    elseif vn==4
        cn='E';
    elseif vn==5
        cn='F';
    elseif vn==6
        cn='X';
    elseif vn==7
        cn='G';
    elseif vn==8
        cn='Y';
    elseif vn==9
        cn='A';
    elseif vn==10
        cn='Z';
    elseif vn==11
        cn='B';
```

```
    end
    %Creamos la cadena de caracteres
    CadenaNotas=strcat(CadenaNotas,cn);
end
end
```

B.3. Representación por intervalos

```
function[CadenaNotas]=ObtenerCadenaIntervalos(ficherotexto,tonalidad,valornegra)
%Leemos el archivo
archivo=textread(ficherotexto,'%s','delimiter','\n');
%Creamos una matriz para quedarnos con la informacion relevante de cada
%linea de texto
M=zeros(length(archivo));
%Guardamos la informacion relevante en la matriz
for i=1:length(archivo)
    s=archivo{i};
    %Guardamos la nota
    for k=1:length(s)
        if s(k)=='='
            t=k;
        end
    end
    M(i)=str2num(s((t+1):(t+3)));
end
%Definimos un vector con los valores de tick de duracion y tonalidad para
%despues definir los vectores para construir las cadenas de notas
%c=[Valor tick negra, tonalidad]
c=[valornegra tonalidad];
%La dimension de la matriz que necesitamos para construir las cadenas de
%notas es de la mitad de filas que la matriz anterior y dos columnas
h=length(M);
l=h/2;
N=zeros(l,1);
for i=1:l
    N(i)=M(2*i)+c(2);
end
%Creamos el vector de intervalos en semitonos
intervalos=zeros(l-1,1);
for i=1:(l-1)
    intervalos(i)=N(i+1)-N(i);
end
%A continuacion calculamos los intervalos diatonicos
diatonicos=zeros(l-1,1);
for i=1:(l-1)
    if intervalos(i)>0
        diatonicos(i)=ceil(intervalos(i)/2);
    elseif intervalos(i)<0
        diatonicos(i)=floor(intervalos(i)/2);
    elseif intervalos(i)==0
        diatonicos(i)=0;
    end
end
%Una vez tenemos la matriz definida, solo queda crear la cadena de
%intervalos
CadenaNotas=' ';
for i=1:(l-1)
    %Definimos el caracter de cada intervalo
    if diatonicos(i)==-12
        cn='a';
```

```
    elseif diatonicos(i)==-11
        cn='b';
    elseif diatonicos(i)==-10
        cn='c';
    elseif diatonicos(i)==-9
        cn='d';
    elseif diatonicos(i)==-8
        cn='e';
    elseif diatonicos(i)==-7
        cn='f';
    elseif diatonicos(i)==-6
        cn='g';
    elseif diatonicos(i)==-5
        cn='h';
    elseif diatonicos(i)==-4
        cn='i';
    elseif diatonicos(i)==-3
        cn='j';
    elseif diatonicos(i)==-2
        cn='k';
    elseif diatonicos(i)==-1
        cn='l';
    elseif diatonicos(i)==0
        cn='m';
    elseif diatonicos(i)==1
        cn='n';
    elseif diatonicos(i)==2
        cn='o';
    elseif diatonicos(i)==3
        cn='p';
    elseif diatonicos(i)==4
        cn='q';
    elseif diatonicos(i)==5
        cn='r';
    elseif diatonicos(i)==6
        cn='s';
    elseif diatonicos(i)==7
        cn='t';
    elseif diatonicos(i)==8
        cn='u';
    elseif diatonicos(i)==9
        cn='v';
    elseif diatonicos(i)==10
        cn='w';
    elseif diatonicos(i)==11
        cn='x';
    elseif diatonicos(i)==12
        cn='y';
    elseif diatonicos(i)==13
        cn='z';
    end
    CadenaNotas=strcat(CadenaNotas,cn);
end
end
```

B.4. Representación por intervalos con reducción

```
function[CadenaNotas]=ObtenerCadenaIntervalosReduccion(ficherotexto,tonalidad,
    valornegra)
%Leemos el archivo
```

```
archivo=textread(ficherotexto,'%s','delimiter','\n');
%Creamos una matriz para quedarnos con la informacion relevante de cada
%linea de texto
M=zeros(length(archivo));
%Guardamos la informacion relevante en la matriz
for i=1:length(archivo)
    s=archivo{i};
    %Guardamos la nota
    for k=1:length(s)
        if s(k)=='='
            t=k;
        end
    end
    M(i)=str2num(s((t+1):(t+3)));
end
%Definimos un vector con los valores de tick de duracion y tonalidad para
%despues definir los vectores para construir las cadenas de notas
%c=[Valor tick negra, tonalidad]
c=[valornegra tonalidad];
%La dimension de la matriz que necesitamos para construir las cadenas de
%notas es de la mitad de filas que la matriz anterior y dos columnas
h=length(M);
l=h/2;
N=zeros(l,1);
for i=1:l
    N(i)=M(2*i)+c(2);
end
%Hacemos la reduccion
%Definimos el contorno de la melodia
contorno=zeros(l,1);
for i=2:l
    if(N(i)-N(i-1))>0
        contorno(i)=1;
    elseif(N(i)-N(i-1))<0
        contorno(i)=-1;
    elseif(N(i)-N(i-1))==0
        contorno(i)=0;
    end
end
%Nos quedamos con las notas repetidas, maximos y minimos locales, la
%primera y la ultima
reduccion=[1];
nnr=1; %numero de notas que entran en la reduccion
for i=2:(l-1)
    if contorno(i)==0
        reduccion=[reduccion i];
        nnr=nnr+1;
    elseif (contorno(i)*contorno(i+1))<=0
        reduccion=[reduccion i];
        nnr=nnr+1;
    end
end
reduccion=[reduccion l];
nnr=nnr+1;
%Creamos un vector con las notas de la reduccion
N1=zeros(nnr,1);
for i=1:nnr
    aux=reduccion(i);
    N1(i)=N(aux);
end
%Creamos el vector de intervalos en semitonos
```

```
intervalos=zeros(nnr-1,1);
for i=1:(nnr-1)
    intervalos(i)=N1(i+1)-N1(i);
end
%A continuacion calculamos los intervalos diatonicos
diatonicos=zeros(nnr-1,1);
for i=1:(nnr-1)
    if intervalos(i)>0
        diatonicos(i)=ceil(intervalos(i)/2);
    elseif intervalos(i)<0
        diatonicos(i)=floor(intervalos(i)/2);
    elseif intervalos(i)==0
        diatonicos(i)=0;
    end
end
%Una vez tenemos la matriz definida, solo queda crear la cadena de
%intervalos
CadenaNotas=' ';
for i=1:(nnr-1)
    %Definimos el caracter de cada intervalo
    if diatonicos(i)==-12
        cn='a';
    elseif diatonicos(i)==-11
        cn='b';
    elseif diatonicos(i)==-10
        cn='c';
    elseif diatonicos(i)==-9
        cn='d';
    elseif diatonicos(i)==-8
        cn='e';
    elseif diatonicos(i)==-7
        cn='f';
    elseif diatonicos(i)==-6
        cn='g';
    elseif diatonicos(i)==-5
        cn='h';
    elseif diatonicos(i)==-4
        cn='i';
    elseif diatonicos(i)==-3
        cn='j';
    elseif diatonicos(i)==-2
        cn='k';
    elseif diatonicos(i)==-1
        cn='l';
    elseif diatonicos(i)==0
        cn='m';
    elseif diatonicos(i)==1
        cn='n';
    elseif diatonicos(i)==2
        cn='o';
    elseif diatonicos(i)==3
        cn='p';
    elseif diatonicos(i)==4
        cn='q';
    elseif diatonicos(i)==5
        cn='r';
    elseif diatonicos(i)==6
        cn='s';
    elseif diatonicos(i)==7
        cn='t';
    elseif diatonicos(i)==8
```

```
        cn='u';
    elseif diatonicos(i)==9
        cn='v';
    elseif diatonicos(i)==10
        cn='w';
    elseif diatonicos(i)==11
        cn='x';
    elseif diatonicos(i)==12
        cn='y';
    elseif diatonicos(i)==13
        cn='z';
    end
    CadenaNotas=strcat(CadenaNotas,cn);
end
end
```

B.5. Representación por notas y duración

```
function[CadenaNotas]=ObtenerCadenaNotasYDuracion(ficherotexto,tonalidad,
    valornegra)
%Leemos el archivo
archivo=textread(ficherotexto,'%s','delimiter','\n');
%Creamos una matriz para quedarnos con la informacion relevante de cada
%linea de texto
M=zeros(length(archivo),2);
%Guardamos la informacion relevante en la matriz
for i=1:length(archivo)
    s=archivo{i};
    %Guardamos el tick
    j=1;
    while s(j)~='C'
        j=j+1;
    end
    M(i,1)=str2num(s(2:(j-1)));
    %Guardamos el valor de nota
    for k=1:length(s)
        if s(k)=='='
            t=k;
        end
    end
    M(i,2)=str2num(s((t+1):(t+3)));
end
%Definimos un vector con los valores de tick de duracion y tonalidad para
%despues definir los vectores para construir las cadenas de notas
%c=[Valor tick negra, tonalidad]
c=[valornegra tonalidad];
%La dimension de la matriz que necesitamos para construir las cadenas de
%notas es de la mitad de filas que la matriz anterior y dos columnas
h=size(M);
l=h(1)/2;
N=zeros(l,2);
%En la primera columna tenemos la duracion de cada nota, y en la segunda la
%nota que es. La duracion va en valores de negra, es decir, si la nota es
%una negra, le asignaremos el valor 1, si es de una corchea 0.5, ... El
%valor de la nota esta ya transportada
for i=1:l
    N(i,1)=M(2*i,1)-M(2*i-1,1);
    N(i,1)=N(i,1)/c(1);
    N(i,2)=M(2*i,2)+c(2);
end
```

```
%Una vez tenemos la matriz definida, solo queda crear la cadena de notas
CadenaNotas='␣';
for i=1:l
    %Si tenemos un tresillo(siempre de corcheas), cambiamos las duraciones
    %de las notas del tresillo corchea semicorchea semicorchea
    if N(i,1)==1/3 && N(i+1,1)==1/3 && N(i+2,1)==1/3
        N(i,1)=1/2;
        N(i+1,1)=1/4;
        N(i+2,1)=1/4;
    end
    %Definimos el caracter de cada nota
    vn=mod(N(i,2),12); %vn=valornota
    if vn==0
        cn='C'; %cn=caracternota
    elseif vn==1
        cn='V';
    elseif vn==2
        cn='D';
    elseif vn==3
        cn='W';
    elseif vn==4
        cn='E';
    elseif vn==5
        cn='F';
    elseif vn==6
        cn='X';
    elseif vn==7
        cn='G';
    elseif vn==8
        cn='Y';
    elseif vn==9
        cn='A';
    elseif vn==10
        cn='Z';
    elseif vn==11
        cn='B';
    end
    %Ahora por cada semicorchea de duracion aadiremos el caracter obtenido
    %antes a la cadena denotas
    ns=N(i,1)/0.25; %ns=numerosemicorcheas
    for j=1:ns
        CadenaNotas=strcat(CadenaNotas,cn);
    end
end
end
```

Apéndice C

Código Edit-Distance (Matlab)

```
function [V,v] = EditDistance(string1,string2)
% Edit Distance is a standard Dynamic Programming problem. Given two strings s1
    and s2, the edit distance between s1 and s2 is the minimum number of
    operations required to convert string s1 to s2. The following operations are
    typically used:
% Replacing one character of string by another character.
% Deleting a character from string
% Adding a character to string
% Example:
% s1='article'
% s2='ardipo'
% EditDistance(s1,s2)
%> 4
% you need to do 4 actions to convert s1 to s2
% replace(t,d) , replace(c,p) , replace(l,o) , delete(e)
% using the other output, you can see the matrix solution to this problem
%
%
% by : Reza Ahmadzadeh (seyedreza_ahmadzadeh@yahoo.com - reza.ahmadzadeh@iit.it)
% 14-11-2012

m=length(string1);
n=length(string2);
v=zeros(m+1,n+1);
for i=1:1:m
    v(i+1,1)=i;
end
for j=1:1:n
    v(1,j+1)=j;
end
for i=1:m
    for j=1:n
        if (string1(i) == string2(j))
            v(i+1,j+1)=v(i,j);
        else
            v(i+1,j+1)=1+min(min(v(i+1,j),v(i,j+1)),v(i,j));
        end
    end
end
V=v(m+1,n+1);
end
```

Apéndice D

Ejemplo de resultados

Veamos los resultados que obtenemos al coger como "query" una canción con 7 canciones en su familia y compararla con las demás, utilizando la representación por cadenas de intervalos sin reducir de las melodías. También mostraremos los vectores Precision y Recall de esa misma canción con la misma representación.

Cuadro D.1: Resultados de comparar una canción con el resto con los valores ordenados de mejor a peor

Canción	Edit-Distance	Smith-Waterman	Needleman-Wunsch
1	1	1	1
2	0.9777	0.8257	0.9837
3	0.9689	0.8154	0.9767
4	0.9496	0.7827	0.9661
5	0.9496	0.7713	0.9654
6	0.9185	0.7483	0.9654
7	0.9130	0.7441	0.9623
8	0.9043	0.7425	0.9618
9	0.9043	0.7320	0.9597
10	0.8966	0.7196	0.9593
⋮	⋮	⋮	⋮
140	0.4458	0.4089	0.3789
141	0.4423	0.4037	0.2451
142	0.4407	0.4037	0.1192
143	0.4360	0.3990	0.0709
144	0.4037	0.3980	0.0642
145	0.3881	0.3922	0.0416
146	0.3876	0.3921	0.0226
147	0.3821	0.3577	0.0191
148	0.2707	0.3358	0.0035
149	0.2707	0.3358	0

Cuadro D.2: Vectores Precision y Recall de la misma canción

Método		Valores del vector																		
Edit-Distance	Precision	0	0	0	0.0357	0.0572	0.0625	0.0597	0.0581	0.0476	0.0410	0.0360	0.0414	0.0408	0.0403	0.0403	0.0403	0.0403	0.0403	0.0403
	Recall	0	0	0	0.1667	0.3333	0.5	0.6667	0.8333	0.8333	0.8333	0.8333	1	1	1	1	1	1	1	1
Smith-Waterman	Precision	0	0	0	0	0	0.05	0.0455	0.0430	0.0407	0.0451	0.0423	0.0408	0.0405	0.0403	0.0403	0.0403	0.0403	0.0403	0.0403
	Recall	0	0	0	0	0	0.3333	0.5	0.6667	0.8333	1	1	1	1	1	1	1	1	1	1
Needleman-Wunsch	Precision	0	0.0286	0.0135	0.0104	0.0093	0.0268	0.0348	0.0427	0.0403	0.0373	0.0365	0.0432	0.0429	0.0429	0.0429	0.0426	0.0426	0.0423	0.0417	0.0403
	Recall	0	0.1667	0.1667	0.1667	0.1667	0.5	0.6667	0.8333	0.8333	0.8333	0.8333	1	1	1	1	1	1	1	1	1