UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

FACULTAD DE INGENIERÍA ELÉCTRICA

División de Estudios de Posgrado

SEGUIMIENTO DE AUDIO MEDIANTE UN ÍNDICE DE PROXIMIDAD

TESIS

Que para obtener el grado de

MAESTRO EN CIENCIAS EN INGENIERÍA ELÉCTRICA

presenta
Luis Fernando Guzmán Nateras

Director de Tesis
José Antonio Camarena Ibarrola
Doctor en Ciencias en Ingeniería Eléctrica

Morelia, Michoacán, México.
Marzo 2014

Resumen

El Seguimiento de Audio (*AF*, del Inglés *Audio Following*) es un proceso mediante el cual se mapea la interpretación de una pieza musical realizada por un músico, usualmente en tiempo real, con una interpretación base utilizada como referencia. Esta interpretación base se considera como "bien interpretada" y es por ello que la interpretación en vivo debería alinearse con ella.

Este tipo de seguimiento tiene muchas aplicaciones como pueden ser el acompañamiento automatizado, la sincronización de efectos visuales o de audio y la enseñanza, entre otros.

En este trabajo se propone el uso de un índice de proximidad para realizar el seguimiento. En particular se utiliza un índice aproximado basado en *Hashing* Sensible a la Localidad (LSH) en conjunto con una firma de audio (AFP, del Inglés *Audio Fingerprint*) que usa como característica de extracción la entropía de la señal. A este enfoque se le denominó *Seguimiento de Audio mediante* (IAF, del Inglés *Index Audio Following*) *Índices*. Para efectuar el seguimiento se extrae la huella de audio de una interpretación que después es indexada usando el índice LSH, esta interpretación se utilizará como interpretación de referencia. Posteriormente, se toman subfirmas equivalentes a medio segundo de audio de una segunda interpretación y se busca su posición correspondiente en la firma de la interpretación de referencia utilizando el índice creado previamente. Así, el enfoque propuesto no utiliza un enfoque de nota a nota puesto que no se mapea a la partitura de la interpretación. En su lugar, se utiliza la firma de audio de una interpretación como referencia para el alineamiento.

Para validar el desempeño se propone un esquema novedoso que consiste en la creación de huellas de audio sintéticas las cuales son modificadas en puntos específicos conocidos con lo cual se conoce de antemano el comportamiento esperado del seguimiento. Se obtuvieron resultados satisfactorios de seguimiento en los experimentos realizados utilizando búsqueda secuencial con el algoritmo propuesto. Al utilizar el índice LSH, se logró una reducción de entre el 60 % y el 75 % de las comparaciones entre firmas realizadas respecto al número de comparaciones hechas al utilizar la búsqueda secuencial, siempre manteniendo un comportamiento equivalente al obtenido con búsqueda secuencial.

Palabras Clave: Seguimiento, Audio, Índices, Procesamiento, Señales

Abstract

Audio Following (AF) is the process of mapping a musician's live performace, usually in realtime, to a base performace that is used as a reference. Said base performace is considered as a "correct performace" and thus, the live performace must align to it.

Audio Following has lots of posible applications such as automatic accompaniment, visual and audio effects synchronization, teaching environments, among others.

This project discusses the use of a proximity index to perform audio following. In particular, we use an approximate search index based in *Locality Sensitive Hashing* (LSH) in conjuction with an audio fingerprint (AFP) that uses the signal's entropy as a feature for extraction. This schema was named *Index Audio Following* (IAF) by the author. To perform the audio following, first the audio fingerprint of one performance is extracted. Then, it is indexed using the LSH index. This performance's AFP will be used as a reference to align the second performance. Afterwards, half-second subfirms of a second performance are extracted and their correspondig positions in the reference AFP are searched using the LSH index. Thus, the proposed schema does not use a note-by-note approach due to the fact that the mapping is not performed againts the performance's score. Instead, the audio fingerprint of a performance is used as a reference for the alignment.

Performace validation is performed using a proposed novel method that consists in the creation of synthetic AFPs that are modified in specific known points so that the expected following behavior is known beforehand. Satisfactory results were obtained with the proposed algorithm when a sequential search was used. When the LSH index was implemented and tested, a reduction of 60 % to 75 % in the number of performed comparisons between AFPs was achieved compared with the number of comparisons performed using a sequential search, while maintaining a performance equivalent to the one obtained through sequential search.

Keywords: Audio, Following, Indexing, Signal, Processing

Contenido

Lista de Figuras

Lista de Tablas

Lista de Algoritmos

Lista de Símbolos

AF	Seguimiento de Audio, *Audio Following*.
SF	Seguimiento de Partitura, *Score Following*.
AA	Alineamiento de Audio, *Audio Alignment*.
AFP	Huella de Audio, *Audio Fingerprint*.
$MIDI$	Interfaz Digital para Instrumentos Musicales. *Musical Instrument Digital Interface*.
$MFCC$	Coeficientes Cepstrales de Frecuencia de Mel. *Mel Frequency Cepstral Coefficients*.
LSH	Hashing Sensible a la Localidad. *Locality Sensitive Hashing*.
HMM	Modelo Oculto de Markov. *Hidden Markov Model*.
$HHMM$	Modelo Oculto de Markov Jerárquico. *Hierarchichal Hidden Markov Model*.
PDF	Función de Densidad de Probabilidad. *Probability Density Function*.
DP	Programación Dinámica. *Dynamic Programming*.
DTW	Doblado Dinámico del Tiempo. *Dynamic Time Warping*.
OTW	Doblado En Línea del Tiempo. *On − line Time Warping*.
IAF	Seguimiento de Audio mediante Índices. *Index Audio Following*.
$\mathbb{R}^b$	Espacio vectorial de dimensión b.
$H^{b'}$	Espacio métrico de Hamming de dimensión b'.
$d(x, y)$	Distancia entre los objetos x y y.
b	Dimensión del espacio.
l	Número de instancias LSH.
n	Número de bits/posiciones utilizados para crear el hash de un vector.
p, q	Elementos de consulta
T	Conjunto de tablas hash.

$mtlf$	Matriz a buscar, *matrix to look for*.
$hash()$	Función hash.
h	Valor hash calculado.
k	Número de vecinos más cercanos.
knn	Lista de vecinos más cercanos.
$distanciasKnn$	Lista de distancias a los vecinos más cercanos.
cp	Posición candidato, *candidate position*.
sp	Posición inicial, *starting position*.
pp	Posición previa o *previous position*.
np	Posición siguiente o *next position*.
LCS	Subsecuencia común más larga, *longest common subsequence*.
pnp	Posición siguiente probable o *probable next position*.
ppp	Posición previa a la anterior o *previous to previous position*.
npp	Predicción de posición siguiente o *next position prediction*.
npt	Posición siguiente en tiempo o *next position in time*.
npd	Posición siguiente en distancia o *next position in distance*.
dt	Distancia reportada por la matriz de npt (*distance time*).
dd	Distancia reportada por la matriz de npd (*distance distance*).
wmd	Parámetro $windowMultiplierDefault$.
nB	Parámetro $numberOfBands$.
hmb	Parámetro $howManyBits$.
hmm	Parámetro $howManyMaps$.
bV	Parámetro $bitVariations$.
ms	milisegundos.

Capítulo 1

Introducción

El procesamiento digital de señales en conjunto con técnicas de búsqueda y recuperación de información, han adquirido mucha importancia en la actualidad debido a la gran cantidad de problemas en los que resulta conveniente su utilización.

Uno de dichos problemas es el Seguimiento de Audio (AF, del Inglés *Audio Following*) el cual consiste básicamente en la alineación de una interpretación en vivo realizada por un músico con una interpretación utilizada como referencia. Un problema similar al AF conocido como Seguimiento de Partitura (SF, del Inglés *Score Following*) ha sido abordado de numerosas maneras en el pasado (**?**), (**?**), (**?**), (**?**), (**?**), (**?**), (**?**), (**?**), (**?**) con buenos resultados.

En este capítulo se presenta una explicación de los problemas AF y SF, una descripción de los diferentes enfoques que se han utilizado para abordarlos previamente y, por último, el esquema propuesto durante este trabajo.

1.1. Planteamiento del Problema

En esta sección se brinda una explicación de dos variantes utilizadas para abordar el problema de seguimiento de audio, las diferencias y similitudes entre ellos, sus aplicaciones y retos. Así mismo, se describe el esquema de seguimiento planteado para este trabajo.

1.1.1. Seguimiento de Partitura

SF es la sincronización en tiempo real de un modelo computacional, como puede ser un HMM (**?**), con un músico que interpreta una partitura musical determinada previamente. El principio detrás del *SF* es bastante simple: introducir una partitura musical a una computadora de alguna forma, y después tocar dicha partitura utilizando la interpretación como una entrada en tiempo real para la computadora, ya sea mediante una interfaz MIDI (Interfaz Digital para Instrumentos Musicales, *Musical Instrument Digital Interface*) o con un micrófono realizando algún tipo de análisis acústico. La meta del sistema es entonces el mapear la secuencia entrante de audio con la representación almacenada de la partitura y determinar en tiempo real la posición del intérprete, usualmente utilizando un enfoque de nota a nota. En la figura **??** se aprecia el funcionamiento de SF, se toma un segmento de audio de la interpretación musical y se busca la posición correspondiente en la partitura.

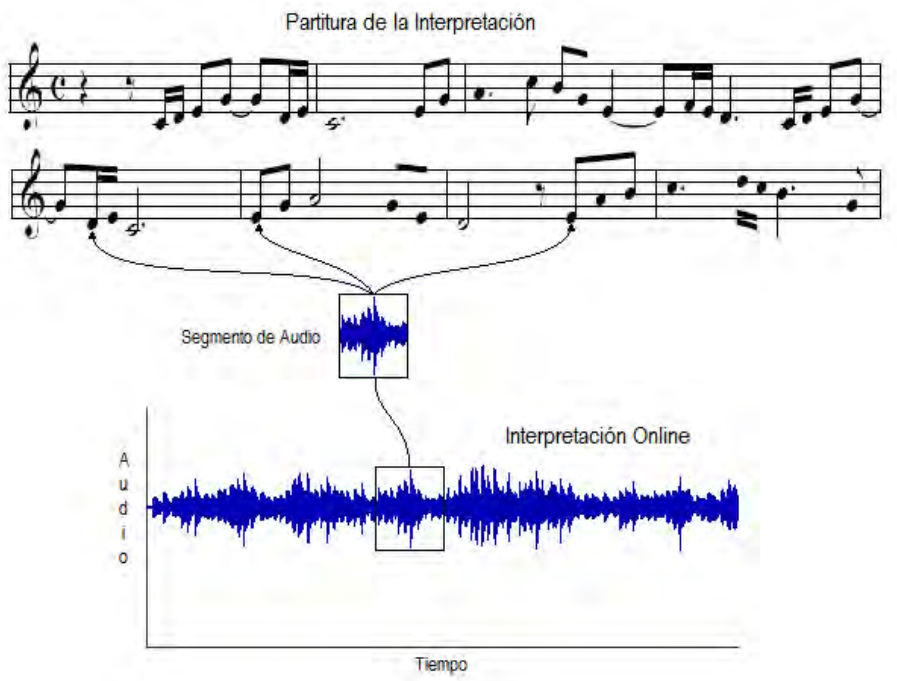


Figura 1.1: Ejemplo de SF.

El *SF* tiene muchas aplicaciones potenciales: se puede utilizar para complementar la experiencia de los asistentes a un concierto (aplicando efectos de sonido en ciertas

secciones, aumentos o disminuciones en el volumen, efectos visuales, pirotecnia, etc.), en contextos de enseñanza/aprendizaje dando retroalimentación a los maestros y estudiantes y sistemas interactivos, entre otros. En resumen, ejecutando eventos electrónicos en puntos precisos de la interpretación.

Uno de los usos más comunes del SF es con el propósito de proveer acompañamiento automatizado para un intérprete en vivo. El acompañamiento en tiempo real resuelve la mayoría de los problemas de sincronización que son inherentes en el acompañamiento grabado. Sin embargo, surgen entonces tres subproblemas: detectar y procesar las entradas producidas por el intérprete, alinear dichas entradas contra la partitura adecuada y generar la información del ritmo necesaria para controlar la generación del acompañamiento.

También es de esperarse que la interpretación en vivo contenga errores que pueden afectar el alineamiento con la partitura esperada. Estos errores pueden ser de dos tipos:

1. **Accidentales:** notas tocadas incorrectamente, notas extra, notas no interpretadas o saltadas, un punto de inicio equivocado, ritmo equivocado, etc.

2. **Provocados:** cambios intencionales, ya sea en el ritmo o en la propia interpretación (saltos, notas adicionales), realizados por el intérprete.

A lo largo de los últimos 30 años se han desarrollado sistemas de acompañamiento automático, todos los cuales utilizan alguna forma de SF (**?**), (**?**), (**?**), (**?**), (**?**), (**?**),(**?**), (**?**).

1.1.2. Alineamiento de Audio y Seguimiento de Audio

El Alineamiento de Audio (AA, del Inglés *Audio Alignment*) y el Seguimiento de Audio (AF, del Inglés *Audio Following*) son problemas hermanos del SF. Ambos consisten en la alineación de dos interpretaciones musicales entre sí en lugar de alinear una interpretación con su partitura como se hace en SF. En general, el AA es el proceso de obtener puntos correspondientes en el tiempo entre dos interpretaciones iguales o con contenidos similares.

En AA se conocen de antemano ambas interpretaciones en su totalidad y la alineación se realiza de manera offline. Por otro lado, en AF se conoce una de las interpretaciones en su totalidad y se utiliza como referencia ,también llamada interpretación base; mientras

que la segunda interpretación, llamada interpretación online, se obtiene paulatinamente. La alineación entre ambas interpretaciones se realiza usualmente en tiempo real conforme se recibe como entrada los datos de la segunda interpretación. En resumen, la diferencia fundamental entre AA y AF consiste en que en AA no se tienen las restricciones adicionales de tiempo real y conocimiento parcial de una de las interpretaciones que usualmente se manejan en el AF. Los resultados del alineamiento offline generado por los sistemas de AA se pueden utilizar como un medio de entrenamiento o comparación de los sistemas en tiempo real AF para validar su desempeño.

Los sistemas AF comparten la mayoría de las aplicaciones y problemas mencionados en la sección **??** para los sistemas de SF con la diferencia evidente de que la alineación se realiza con una interpretación de referencia en lugar de con la partitura. En la figura **??** se observa el funcionamiento del AF, se toma un segmento de audio de la interpretación online y se busca su posición correspondiente en la interpretación de referencia.

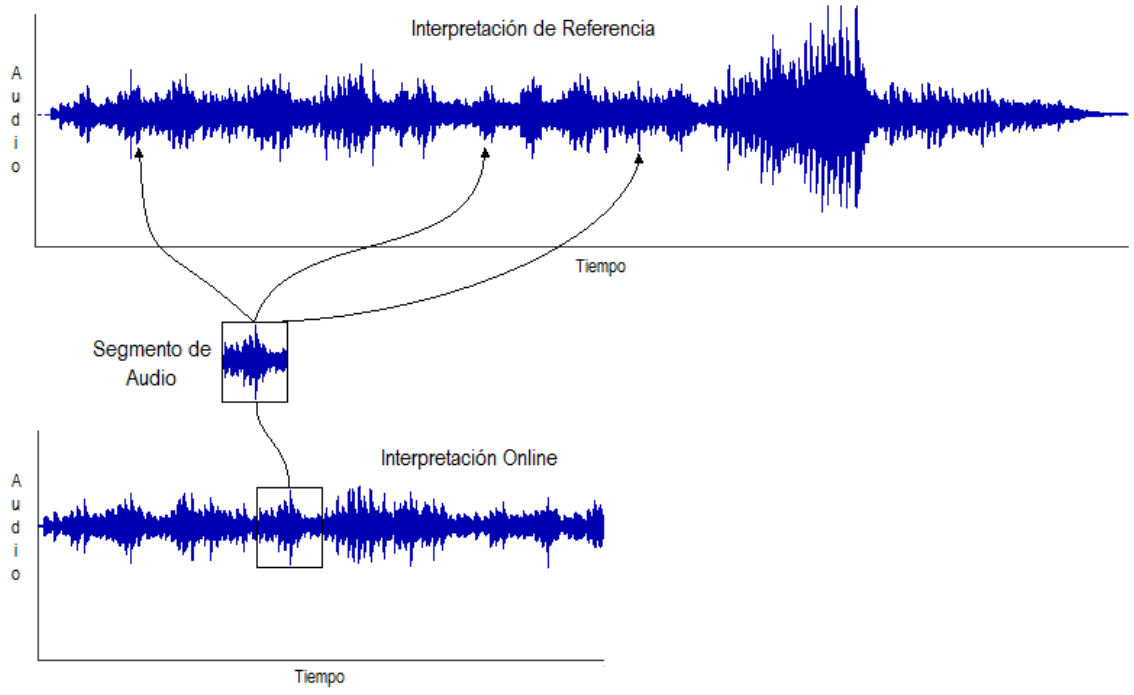


Figura 1.2: Ejemplo de AF.

1.1.3. Esquema de Seguimiento Propuesto

En este trabajo se plantea la implementación de un sistema de *AF*. Utilizando un par de interpretaciones de una misma pieza musical, se utilizará una de ellas como interpretación base o de referencia y la otra como interpretación online. Se obtendrá un segmento de la interpretación online y se buscará su posición correspondiente dentro de la interpretación base.

Para poder encontrar la posición correspondiente del segmento de audio de la interpretación online dentro de la interpretación de referencia, es necesario comparar de manera secuencial dicho segmento con todos los segmentos de audio equivalentes de la interpretación base como se muestra en la figura **??**. A este esquema se le conoce como búsqueda secuencial.

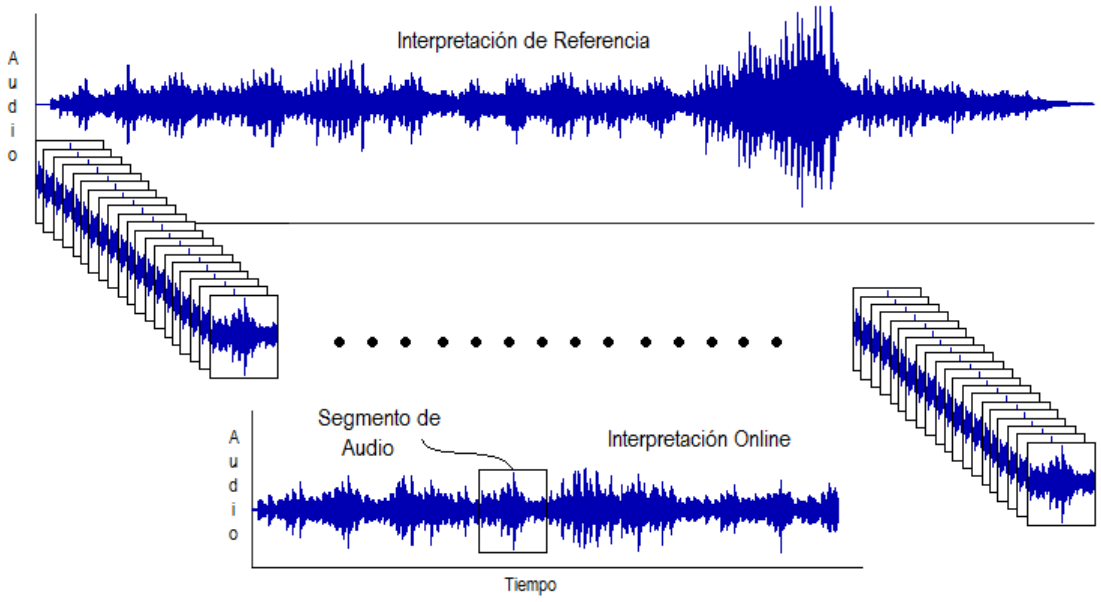


Figura 1.3: Ejemplo de búsqueda secuencial.

Para facilitar las comparaciones entre segmentos de audio, usualmente se realiza algún tipo de extracción de características de la señal de audio de manera que ésta pueda ser representada de forma adecuada por los vectores de características extraídos. A esta representación se le conoce como Firma o Huella de Audio (AFP, del Inglés *Audio Fingerprint*).

Diferentes AFPs utilizan distintas características de extracción como pueden ser el volumen, tono, ritmo, nota, energía (**?**), entropía, llanura espectral (**?**), etc (**?**)(**?**). De esta manera en lugar de comparar los segmentos de audio directamente se comparan segmentos de las AFP obtenidas a los cuales se les conoce como subfirmas de audio como se presenta en la figura **??**.

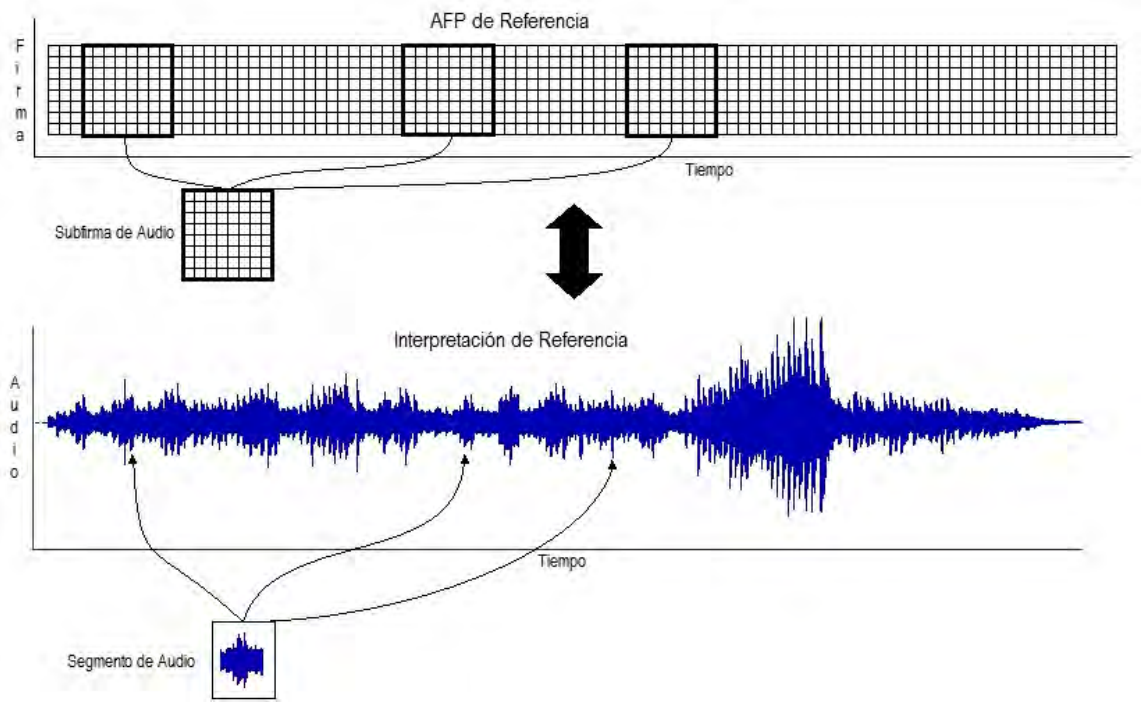


Figura 1.4: Conversión de audio a AFP.

Dependiendo de la AFP utilizada la comparación entre subfirmas se puede realizar utilizando diferentes funciones de distancia. Por ejemplo, si la AFP entrega vectores de características compuestos por números reales se puede utilizar la distancia Euclidiana (**?**) o la distancia Manhattan (**?**). En cambio si la AFP utiliza algún tipo de codificación de los vectores, por ejemplo a cadenas de caracteres, se pueden utilizar las distancias de Hamming (**?**) o Levenshtein (**?**).

En particular, para el presente trabajo se utiliza la AFP publicada en (**?**) la cual utiliza como característica de extracción la entropía de la señal y realiza una codificación binaria de los vectores de características por lo cual para realizar las comparaciones entre

subfirmas se utilizaron las distancias de Hamming, Levenshtein y LCS (Subsecuencia común más larga, del Inglés *Longest Common Subsequence*) (**?**).

Es claro que al utilizar la búsqueda secuencial descrita anteriormente se realiza una gran cantidad de comparaciones entre subfirmas, muchas de ellas innecesarias puesto que se realizan entre subfirmas muy diferentes. Para evitar el uso de la búsqueda secuencial y reducir el número de comparaciones entre subfirmas efectuadas, se propone la utilización de un índice de proximidad. La AFP de la interpretación de referencia es indexada utilizando el índice de proximidad y cuando se busca encontrar las posibles posiciones de una subfirma de la interpretación online se realiza una consulta al índice la cual entregará como respuesta únicamente las posiciones de las subfirmas de la interpretación de referencia que sean más similares a la subfirma de consulta. Lo anterior se ejemplifica en la figura **??**

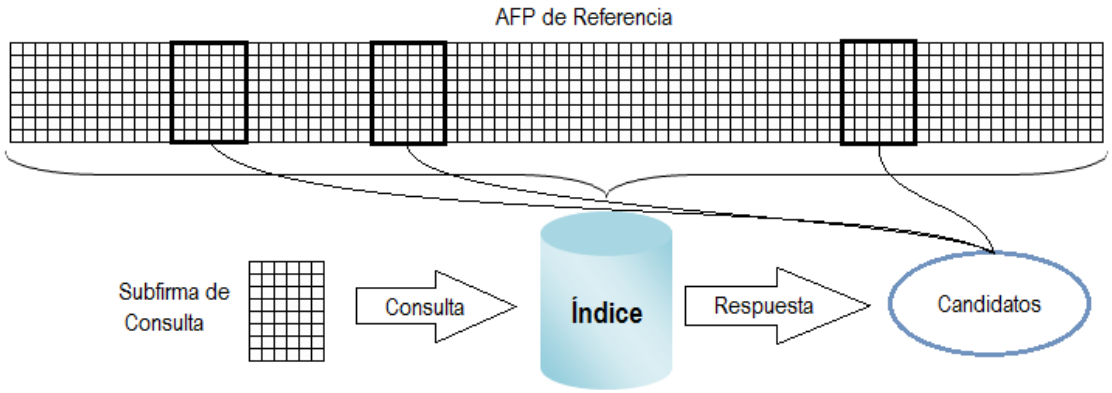


Figura 1.5: Consulta de una subfirma en el índice.

Como en el caso de las AFPs, también existen distintos índices de proximidad que se pueden utilizar. Para este trabajo, debido a las características de los vectores entregados por la AFP elegida, se decidió utilizar el índice de proximidad basado en *Hashing* Sensible a la Localidad (LSH, del Inglés *Locality Sensitive Hashing*) presentado por (**?**).

Puesto que el esquema utilizado por el presente trabajo es distinto al utilizado por la mayoría de los trabajos relacionados previamente (no se utiliza un enfoque de nota-a-nota) es difícil efectuar una comparación, por ello, para validar el desempeño del seguidor implementado se propone la utilización de un esquema novedoso el cual consiste en la

creación de firmas de audio sintéticas. Estas firmas de audio sintéticas son modificadas en puntos específicos conocidos de manera que se conoce de antemano el comportamiento esperado del seguimiento y es posible comparar el seguimiento obtenido con el esperado. Adicionalmente, se obtuvieron manualmente las mediciones de los momentos de ejecución de las notas (*note onset*) de algunas de las interpretaciones utilizadas durante los experimentos y de esta manera se puede conocer la alineación real de dichas interpretaciones y compararla con la alineación obtenida por el seguidor.

Al enfoque presentado en este trabajo en el cual se utiliza un índice de proximidad para realizar la búsqueda se le denominó Seguimiento de Audio mediante un Índice de Proximidad (IAF, del Inglés $IndexAudioFollowing$) para diferenciarlo de otras versiones de AF.

1.2. Antecedentes y Estado del Arte

El problema de SF no es algo nuevo, fue presentado por primera vez en la Conferencia Internacional de Música por Computadora ($ICMC$, $International\ Computer\ Music\ Conference$) de 1984 por Barry Vercoe y Roger Dannenburg de manera independiente. En un inicio, el objetivo fue poder proveer a los intérpretes con un sistema capaz de realizar acompañamiento automático.

En (**?**) se describe el área de investigación por primera vez de la siguiente manera: "...entender la dinámica de una interpretación en conjunto en vivo lo suficientemente bien como para poder reemplazar a cualquier miembro del grupo con un intérprete sintético (por ejemplo, un modelo de computadora) de manera que el resto de los integrantes no puedan notar la diferencia...". En la implementación de Vercoe, se emplearon interpretaciones del flautista profesional Larry Beauregard y se propuso la utilización del tono (*pitch*) como característica de extracción. Vercoe menciona que para obtener y detectar una secuencia de eventos tocados por un flautista profesional es necesario realizar la detección del tono a una velocidad tal que es casi imposible para los métodos acústicos. Por ello plantea además la utilización de dos fuentes adicionales de datos: la partitura musical y el tiempo de ejecución de las notas (*onset*). Para ello se utilizó una flauta a la cual se le instalaron sensores ópticos

en las teclas con el fin de obtener las mediciones tiempo de ejecución de las notas.

En (**?**) se presenta un algoritmo eficiente de programación dinámica para encontrar la mejor concordancia entre una interpretación y su partitura. Este algoritmo compara la secuencia de eventos generados por el intérprete con la secuencia de eventos esperados. Para encontrar la mejor concordancia, se calcula una matriz de enteros donde cada fila corresponde a un evento en la partitura y cada columna corresponde a un evento detectado en la interpretación. Cada vez que se detecta un nuevo evento dentro de la interpretación se calcula una nueva columna. El entero calculado para la fila r y la columna c responde a la pregunta: Si nos encontráramos en el evento r-ésimo evento de la partitura y el c-ésimo evento de la interpretación, ¿cuál sería la longitud de la concordancia entre los eventos esperados y los eventos obtenidos hasta este momento? De la misma manera que el trabajo presentado en (**?**), el trabajo de Dannenberg es orientado a la generación de un acompañamiento automático en tiempo real. Una de las limitaciones del algoritmo es que únicamente trata con música monofónica y un sólo instrumento.

Posteriormente en (**?**) y (**?**) se describió un nuevo enfoque que utiliza funciones de densidad de probabilidad. Dada una determinada posición previa en la partitura, el algoritmo de seguimiento continuamente estima la distancia desde dicha posición. Entonces se utilizan las observaciones más recientes del tono, picos espectrales, cambios en la amplitud, etc. para ubicar la nueva posición del intérprete.

Los algoritmos de Vercoe y Dannenberg fueron extendidos para poder tratar con música polifónica e instrumentos múltiples en (**?**).

En (**?**) se presenta un resumen del algoritmo usado hasta esa fecha en el Instituto de Investigación y Coordinación Acústica Musical (*IRCAM*, *Institut de Recherche et Coordination Acoustique/Musique*) para realizar acompañamiento automático. Dicho algoritmo utiliza un enfoque de nota por nota y utiliza como característica de extracción el tono (*pitch*) (**?**) de la interpretación. Utiliza punteros a la "nota actual" y una lista de notas anteriores llamadas "notas saltadas" (*skipped notes*). Cuando se toca una nota, el algoritmo intenta alinearla con alguna nota en la base de datos; la nota alineada debe tener el mismo tono que la interpretada en vivo (aunque se aceptan notas con diferencia de una octava). Primero se busca una coincidencia dentro de la lista de notas saltadas, si no se

encuentra ahí, la búsqueda se continúa a partir de la nota actual. Cuando se encuentra una coincidencia, se actualiza el puntero hacia la nota actual y se agregan las nuevas notas saltadas a la lista. Una vez que la última nota se ha alineado, el algoritmo termina.

En (**?**) se propone por primera vez utilizar un modelado estocástico del problema para lidiar con los problemas de incertidumbre asociados con el uso de la voz (en la cual las señales obtenidas no asemejan para nada a una secuencia de tonos discretos en intervalos determinados) y los posibles errores cometidos por los intérpretes instrumentales. Se propone la extracción de seis diferentes características de la señal: la energía, el cambio de la energía entre un marco y otro, los cruces por cero, la frecuencia fundamental (**?**), el cambio de la frecuencia fundamental entre un marco y otro y el error de la frecuencia fundamental. Estas características se utilizan para crear una secuencia de observaciones la cual será introducida a un HMM y se utiliza el algoritmo de Viterbi (**?**) para encontrar la secuencia de estados más probable producida por el modelo. Se utilizan tres HMMs de izquierda a derecha ($left\text{-}to\text{-}rigth$) para modelar la presencia de una nota, la ausencia de nota y los silencios. El modelo utilizado para las notas cuenta con tres estados que representan el comportamiento habitual de una nota: ejecución (*attack*, *onset*), mantenimiento (*sustain*, *hold*) y fin (*release*). El modelo del para los silencios únicamente cuenta con un estado puesto que los silencios no cuentan con una estructura. El modelo de ausencia de nota cuenta también con tres estados y pretende representar todos los sonidos dentro de la interpretación que no son considerados como notas, éstos pueden ser ruidos o, en el caso de la voz, las aspiraciones, los sonidos fricativos o plosivos.

En (**?**) se trata con un subproblema del seguimiento de audio: la segmentación de las señales acústicas. También hace uso de HMMs como modelo. Dada la partitura de una pieza de música monofónica y la grabación de una interpretación de dicha pieza, se segmenta la señal en regiones contiguas correspondientes a las notas y los descansos con el fin de poder estimar la posición de la interpretación dentro de la partitura y proveer acompañamiento automático.

En (**?**) se propone un esquema similar al utilizado en (**?**) pero utiliza un HMM de dos niveles: el nivel inferior compara las características de la señal de entrada con las esperadas; el nivel superior se utiliza para modelar la interpretación como una secuencia

de eventos musicales tomando en consideración los posibles errores que puede cometer el intérprete. El HMM del nivel superior utilizado para modelar la interpretación tiene dos tipos de estados: estados normales llamados *n-states* y estados fantasma *g-states*. Los estados normales corresponden a eventos interpretados correctamente mientras que los estados fantasma corresponden a los posibles errores locales entre los eventos esperados en la partitura y los obtenidos en la interpretación. Cada evento es representado por un *n-state* y un *g-state* paralelo. La topología del HMM es de izquierda a derecha debido a la precedencia temporal de los eventos. Cada *n-state* está conectado hacia los subsecuentes estados n y g. Las transiciones de los *g-states* toman en cuenta tres posibles tipos de errores: notas equivocadas, notas extra y notas saltadas. El HMM del nivel inferior se utiliza para modelar la señal de entrada. Cada estado del nivel superior se compone por un conjunto de estados del nivel inferior. Estos estados toman en cuenta, por cada evento, las características relacionadas con la ejecución, mantenimiento y posible silencio al final. Adicionalmente se presenta un algoritmo alternativo a Viterbi para determinar la secuencia de estados más probable, el cual maximiza la probabilidad de alineamientos locales en lugar de maximizar la alineación global.

En (**?**) se expone una metodología para realizar el alineamiento offline de interpretaciones musicales monofónicas y polifónicas. Se basa en el uso del Doblado Dinámico del Tiempo (DTW, del Inglés *Dynamic Time Warping*) (**?**) para calcular la distancia entre los marcos de la interpretación y secciones de la partitura utilizando los picos espectrales de la señal.

En (**?**) se presenta una descripción del estado del arte hasta esa fecha incluyendo el sistema de seguimiento de audio utilizado en el $IRCAM$, el cual, para ese entonces, ya se basaba en HMM.

En (**?**) y (**?**) se propone un esquema para buscar y alinear grabaciones de audio polifónico con las representaciones simbólicas de sus partituras guardadas en archivos MIDI. El método se basa en el uso de *Chromagramas* (**?**) y DTW. Los chromagramas son secuencias de vectores de *chromas*. Los vectores chromas son vectores de 12 elementos donde cada elemento representa la energía espectral correspondiente a un tono (por ejemplo, Do, Do#, Re, Re#, Mi, etc.). Se convierten los archivos MIDI a audio utilizando un sintetiza-

dor y posteriormente se convierten ambos archivos de audio a vectores de chromas. Estas secuencias se alinean utilizando DTW. Para validar su esquema de alineación proponen alinear interpretaciones musicales con archivos MIDI que no les corresponden y observar gráficamente como el camino óptimo obtenido por DTW no es una diagonal sino que tiene un comportamiento errático. Adicionalmente, introducen cambios artificiales en los archivos MIDI de manera que las alineaciones obtenidas se ven afectadas por dichos cambios.

En (**?**) se presenta una alternativa para realizar el seguimiento de una interpretación en vivo utilizando un método conocido como *Online Time Warping* (OTW). El DTW no resulta adecuado para aplicaciones en línea debido a que tiene la limitación de requerir conocimiento por completo de ambas series de tiempo para poder calcular su alineamiento. *OTW* realiza alineamientos incrementales de dos series de tiempo mientras que una de ellas se recibe en tiempo real. Dicho algoritmo se aplica al alineamiento de señales de audio para poder dar seguimiento a interpretaciones musicales de longitudes arbitrarias. Esta implementación utiliza una representación espectral de los datos del audio. El resultado del trabajo presentado por Dixon se compiló en un *toolkit* para el alineamiento de grabaciones de audio llamado MATCH (**?**). Este *toolkit* puede ser conseguido en línea en (**?**) y se desarrolló un *plug-in* para su uso en conjunto con *Sonic Visualizer* (**?**).

En (**?**) se utiliza un HMM Jerárquico (*HHMM*, *Hierarchichal Hidden Markov Model* (**?**)) de dos jerarquías: el primer nivel consiste en estados musicales de alto nivel correspondientes a las partes atómicas de la partitura: notas, acordes, y descansos. El segundo nivel contiene un modelado temporal de acuerdo al tiempo de ocurrencia de cada evento de la partitura. Para la representación de los datos de audio se utiliza un algoritmo de análisisde tonos múltiples (**?**).

En (**?**) se plantea la solución de un problema alterno pero similar: el seguimiento en tiempo real del ritmo de la interpretación. Inicialmente proponen el uso del filtro de Kalman para calcular un estimación óptima del ritmo usando las predicciones generadas por el modelo y las mediciones tomadas de una interpretación. Posteriormente, el usar un filtro de partículas les permite efectuar el seguimiento del ritmo de una interpretación de la cual se desconoce la partitura. Puesto que para calcular el ritmo de una interpretación es necesario saber o estimar la frecuencia de ocurrencia de las notas, el trabajo presentado

por Kasteren se puede considerar también como seguimiento de audio.

En (**?**) y (**?**) se proponen esquemas que estiman continuamente el ritmo (*tempo*) de la interpretación y lo toman en cuenta para efectuar la alineación. Lo anterior, puesto que en muchas interpretaciones el ritmo no es constante sino que cambia con el tiempo. Incluso si el ritmo se considera constante para cierta pieza musical, los intérpretes humanos son incapaces de mantenerlo y siempre existen variaciones. Además, estos esquemas basados en el ritmo se basan en la idea de que la percepción de la estructura musical es una actividad continua en la cual las expectativas futuras del oyente pueden ser tan importantes como los eventos musicales en sí mismos. En (**?**) se presenta un sistema de alineamiento offline de audio polifónico compuesto por dos etapas consecutivas para estimar la posición y el ritmo. La primera etapa se compone por un HMM derivado de la partitura de la interpretación. Este modelo se encarga de estimar la posición. La segunda etapa utiliza una red bayesiana (**?**) para calcular el ritmo a lo largo de la interpretación. Por el contrario, en (**?**) se centran en el procesamiento online en tiempo real y las estimaciones de ritmo y posición se realizan en paralelo.

En (**?**) se utilizó un esquema de seguimiento para el acompañamiento automático basado en el trabajo de (**?**) usando HMM's de dos niveles para representar los eventos esperados y las posibles desviaciones/errores que pueden tener lugar. Únicamente permite 13 posibles observaciones correspondientes a las 12 notas de la escala cromática más los silencios. Utiliza también el enfoque de nota a nota y para la extracción de características de la señal se utiliza el tono. Una cuestión notable es que, además del tono de la nota, toma en cuenta el volumen de la misma para tratar de que el acompañamiento se toque con un volumen relativo al utilizado por el intérprete.

En (**?**) se presenta un estudio y evaluación de las características de extracción utilizadas en los esquemas de AA y SF con el fin de determinar cuáles entregan los mejores resultados. Se analiza una gran cantidad de características de la señal: la energía, valor máximo, volúmen, espectro, llanura espectral, Coeficientes Cepstrales de Frecuencia de Mel (MFCC, del Inglés $Mel\ Frequency\ Cepstral\ Coefficients$) (**?**), chromas, tiempos de ejecución, entre otros. En los resultados describen que la elección de las características de extracción debe depender de las interpretaciones de audio con las que se trabaje, sin embar-

go se menciona que, si bien se pueden obtener alineamientos precisos utilizando únicamente una característica de extracción, el utilizar múltiples características usualmente arroja mejores resultados. Más aún, el aplicar diferentes pesos a cada una de las características usadas puede en algunos casos mejorar los resultados.

Por último en (**?**) se presenta un sistema de seguimiento musical en tiempo real que puede alinear una interpretación con su correspondiente partitura. Este trabajo describe la implementación una aplicación desarrollada por los autores en la cual los usuarios utilizan un teclado conectado mediante una interfaz MIDI para interpretar una pieza musical cuya partitura se ha cargado previamente en el sistema mediante un archivo MIDI y una imagen. Así, cuando el usuario interpreta una nota en el teclado, se lee la entrada MIDI, se compara con el archivo MIDI de la partitura, se estima la posición del usuario dentro de la partitura y se despliega en la imagen de la partitura un indicador que muestra al usuario su posición. Con este enfoque no se requiere procesamiento de la señal de audio puesto que es convertida a entradas MIDI directamente. Se utiliza el algoritmo de programación dinámica presentado en (**?**) pero extiende dicho algoritmo para poder procesar música polifónica: agrupando y clasificando las notas polifónicas en notas iniciales y notas seguidoras. Adicionalmente, toma en consideración los hábitos del intérprete y las circunstancias, como el repetir partes de la partitura o tocar la nota incorrecta.

Como se mencionó en la sección **??**, para el presente trabajo se utiliza un enfoque que no ha sido considerado hasta la fecha para abordar el problema de AF y AA. Se utiliza la entropía de la señal (**?**) como característica de extracción y se utiliza un índice de proximidad para estimar la posición del intérprete. Estas diferencias hacen que una comparación directa con alguno de los esquemas previamente implementados sea complicada. Por esta razón, se propone una técnica de validación similar a la utlizada en (**?**) en donde se modificaron artificialmente los archivos MIDI utilizados para evaluar el alineamiento obtenido. Se crearán firmas de audio modificadas artificialmente en puntos específicos conocidos de manera que se conozca de antemano el alineamiento esperado entre las interpretaciones y se pueda comparar con el alineamiento obtenido. En conjunto con la técnica de validación mencionada, se obtuvieron manualmente las mediciones de los momentos de ejecución de las notas (*note onset*) de algunas de las interpretaciones utilizadas durante los experimentos y

de esta manera se puede conocer la alineación real de dichas interpretaciones y compararla con la alineación obtenida por el seguidor. El conjunto de interpretaciones utilizadas para efectuar los experimentos de este trabajo es el mismo conjunto utilizado en algunos trabajos previos como (**?**), (**?**) y (**?**).

1.3. Objetivos de la Tesis

1.3.1. Objetivo general

Implementar un esquema para realizar seguimiento de audio utilizando una huella de audio basada en la entropía de la señal y el índice de búsqueda aproximada *Locality Sensitive Hashing*.

1.3.2. Objetivos particulares

- Obtener las huellas de audio de varias interpretaciones utilizando una firma basada en la entropía.
- Diseñar un esquema de seguimiento mediante la búsqueda secuencial de subfirmas dentro de las firmas extraídas previamente.
- Implementar el índice LSH.
- Incrementar la calidad del desempeño del índice LSH usando múltiples instancias.
- Comprimir el índice LSH mediante bitmaps.
- Implementar el esquema *Index Audio Following* o Seguimiento de Audio mediante Índices utilizando las herramientas anteriores.

1.4. Descripción de Capítulos

Este capítulo introduce al lector al proyecto y provee una apreciación global de sus antecedentes, objetivos y contenidos. El resto del documento guarda la estructura presentada a continuación.

En el Capítulo 2 se incluyen los aspectos teóricos de las herramientas utilizadas en la implementación del proyecto: la huella de audio y el índice aproximado *Locality Sensitive Hashing* (LSH).

En el Capítulo 3 se presenta una descripción a detalle de las decisiones de diseño e implementación del esquema de seguimiento de audio planteado, al cual se denominó *Index Audio Following* (*IAF*) o *Seguimiento de Audio mediante Índices*.

El Capítulo 4 expone los experimentos planteados con el fin de validar el funcionamiento del proyecto y los resultados obtenidos al realizar las pruebas de seguimiento; primero mediante el uso de una búsqueda secuencial y posteriormente con el índice LSH implementado. Acompañados de una discusión sobre su significado y desempeño.

Por último en el Capítulo 5 se proporcionan las conclusiones alcanzadas al término del proyecto y se plantean algunos de los posibles trabajos futuros que pueden surgir de la presente investigación.

Capítulo 2

Marco Teórico

En este capítulo se presentan los conceptos teóricos de las herramientas utilizadas para el proyecto.

2.1. Extracción de Características

El primer paso en la identificación de señales de audio consiste en la extracción de las características de dicha señal de manera que éstas representen al audio de manera adecuada. A esta representación generalmente se le conoce como Huella de Audio o Firma de Audio (*Audio Fingerprint, AFP*). Existen diferentes características acústicas que se pueden extraer y analizar como: volúmen, nota, tono, ritmo/compás, etc(**?**)(**?**). Se han diseñado diversas huellas de audio, cada una de las cuales tiene sus características particulares con ventajas y desventajas, pero todas siguen un proceso similar:

El primer paso usualmente es el preprocesamiento, durante el cual, se realizan operaciones como la conversión de la señal a un formato monoaural mediante la promediación de los canales de la señal original.

En seguida se divide la señal en Marcos(*Frames*), pequeños segmentos de la señal (usualmente de entre 50 a 300 ms) y que generalmente tienen un traslape (50 % - 98 %) entre ellos.

Posteriormente se aplica algún tipo de transformación a la señal: la transformada de Fourier, para cambiar al domino de la frecuencia, es utilizada comúnmente pero se puede

utilizar otro tipo de transformación (ej. transformada coseno) dependiendo de la huella que se requiera.

Una vez que se tiene la señal en el dominio deseado, se realiza la extracción de características como pueden ser los espectrogramas, los coeficientes cepstrales de frecuencia de Mel (*Mel Frecuency Cepstral Coefficients, MFCC*), los coeficientes de predicción lineal (*Linear Prediction Coefficients, LPC*), la entropía de la señal, etc.

Opcionalmente se puede realizar un postprocesamiento de la señal, el cuál puede consistir en la cuantización (*clustering*), normalización o alguna operación adicional sobre los vectores de características. Y por último, cada firma elige una representación para sus características las cuales pueden ser vectores, cadenas, valores únicos, etc.

En la Figura **??** describe el proceso de extracción de características.

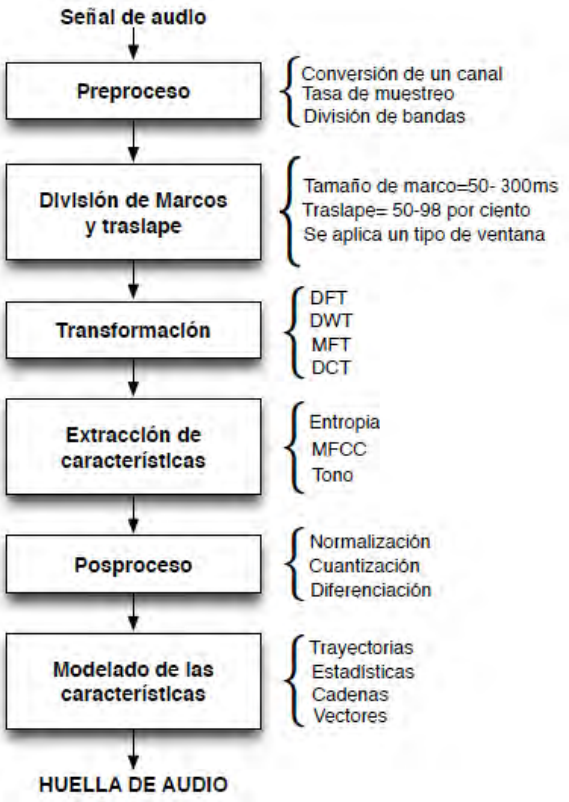


Figura 2.1: Proceso de extracción de características.

Como se mencionó anteriormente, cada huella de audio tiene sus fortalezas y de-

bilidades particulares, lo cual conlleva a que su desempeño dependa en gran medida de las características del problema en cuestión.

Para el presente trabajo se decidió utilizar la firma basada en la entropía publicada en (**?**). Esta firma calcula la entropía de la señal por banda crítica de Bark para cada marco, obteniendo así un vector de 24 valores de entropía. Finalmente realiza una codificación de dichos vectores a 0's y 1's.

Si la entropía de la señal aumentó para una determinada banda respecto al marco anterior se utiliza un 1 y si bajó o se mantuvo igual se utiliza un 0. En la Figura **??** podemos observar el resultado de la codificación utilizada por la huella.

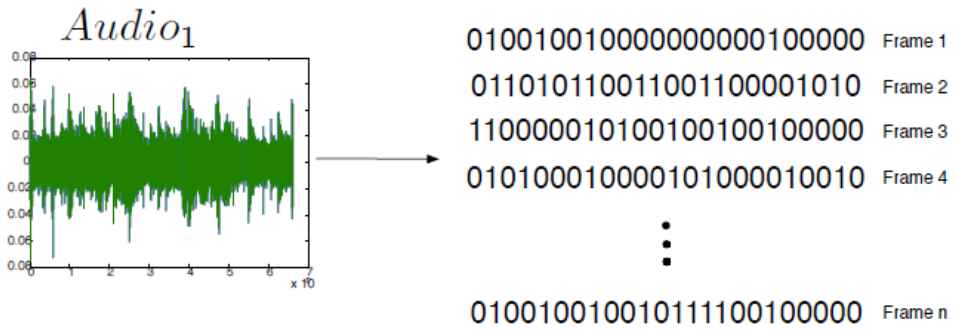


Figura 2.2: Ejemplo de la codificación de la huella de audio.

Debido a las características de esta huella, es altamente compatible para ser utilizada con índices basados en Hashing Sensible a la Localidad (*Locality Sensitive Hashing, LSH*) y se ha utilizado en aplicaciones de recuperación de información como se presentó en (**?**) con un desempeño adecuado.

2.2. Índice LSH

Una manera de clasificar los índices de búsqueda (de entre las muchas que hay) es a partir del tipo de respuesta que garantizan:

1. Los índices con respuesta exacta garantizan que dentro de los resultados encontrados siempre se encontrará al elemento más cercano a la búsqueda.

2. Los índices con respuesta aproximada no garantizan que se entregue al vecino más cercano, pero si resultados muy cercanos a la consulta: tal vez se encuentre al vecino más cercano, tal vez el segundo, tal vez el tercero o cuarto, etc. en general un conjunto de los elementos más cercanos.

Ambos tipos tienen sus ventajas y desventajas. La mayoría de los métodos de búsqueda exacta sufren del fenómeno conocido como la *maldición de la dimensión* como se describe en (**?**). Por otro lado, los métodos que devuelven resultados aproximados intercambian la exactitud por velocidad de respuesta o el espacio de memoria necesario.

La calidad de los resultados entregados por una búsqueda realizada en un índice se determina por la proximidad de los elementos obtenidos respecto al elemento de consulta. Si se entregan los vecinos más cercanos al elemento de consulta, los resultados se consideran de buena calidad. Si por otra parte, los elementos obtenidos por la búsqueda son muy diferentes al elemento de consulta, se considera que los resultados son de mala calidad.

El problema de la búsqueda aproximada es de gran importancia en gran variedad de aplicaciones como pueden ser la compresión y minería de datos, recuperación de información, reconocimiento de patrones, estadística y análisis de datos entre muchos otros. En muchas de estas aplicaciones, no es necesaria la respuesta exacta para una búsqueda, determinar una respuesta aproximada suele ser suficiente.

El Locality Sensitive Hashing fue presentado en (**?**). Es una técnica de búsqueda por proximidad muy rápida que proveé garantías probabilísticas sobre la calidad de los resultados. La idea es aplicar una función hash a los diferentes elementos de la base de datos de manera que se garantice que la probabilidad de que dos elementos obtengan el mismo hash sea mayor para elementos similares que para aquellos que son diferentes.

Usualmente las características de los vectores son representadas como puntos en $\mathbb{R}^b$, donde b es el número de coordenadas del objeto vector y una distancia métrica es utilizada para medir la similitud entre dos objetos.

Gionis propone que para explicar cómo se define la función hash usada en LSH resulta muy adecuado utilizar una representación unaria de los vectores. Para ello se puede realizar un conversión la cual es explicada a detalle en (**?**).

Puesto que la huella de audio utilizada para la extracción de características utiliza una codificación en ceros y unos de los vectores como se explicó en la Sección **??**, la transformación mencionada en el párrafo anterior no es necesaria para los vectores usados en este trabajo.

Usualmente, los índices LSH se implementan como tablas hash de manera que exista una alta probabilidad que los objetos que sean cercanos respecto a alguna función de distancia $d(-,-)$ se encuentren dentro del mismo bucket.

Como se mencionó, LSH ofrece algunas garantías probabilísticas sobre la calidad de los resultados, en resumen, existen límites de distancia entre objetos con el mismo hash. Ésto puede ser suficiente para muchas aplicaciones, sin embargo, existe un límite en la calidad que puede ser garantizada usando una única instancia LSH.

Para aumentar la calidad de los resultados, la solución más simple es utilizar varias instancias. No obstante, lo anterior aumenta los requerimientos de memoria. Por ello se han desarrollado esquemas de compresión para índices LSH como los presentados en (**?**) y (**?**). Particularmente en este trabajo se utilizó un índice LSH comprimido mediante la utilización de los *SArrays* descritos en (**?**) como se presentó en (**?**).

La función hash se define como sigue. Se elige un número determinado de instancias l y un número n de posiciones donde $n <= b$. Para cada instancia l_i se elige un conjunto de n posiciones al azar de entre $1, 2, ..., b$. Si definimos p_{l_i} como la proyección del vector p en el conjunto de posiciones l_i, calculamos p_{l_i} concatenando los bits que se encuentran en las posiciones indicadas en l_i. Denotamos que el *bucket* $g_i(p) = p_{l_i}$. Así, se almacena cada $p \in P$ en el *cubo* $g_i(p)$ para $i = 1, 2, ..., l$ (**?**).

En la Figura **??** se muestra un ejemplo del procedimiento descrito en el párrafo anterior. Se determinó usar 2 instancias ($l = 2$) y utilizar 3 bits ($n = 3$) de los 5 disponibles ($b = 5$) para calcular la función hash. Los bits a utilizar para la instancia 1 ($n1$) son $0, 2, 4$; mientras que para la instancia 2 ($n2$) son $2, 3, 4$. El procedimiento se aplica para cada uno de los vectores de la firma a indexar, sin embargo para fines de este ejemplo se tomó el vector de la posición 4 (00111) como muestra. Para calcular de el hash de la instancia 1 se toman los bits 0,2 y 4 por lo cual el hash obtenido es 011, entonces se agrega la posición del vector (4) a la lista de posiciones del cubo 011 de la primera tabla hash. Así mismo, para

calcular el hash de la instancia 2 se toman los bits 2,3 y 4 por lo cual el hash que se obtiene es 111, entonces se agrega la posición 4 a la lista de posiciones del cubo 111 de la tabla 2.

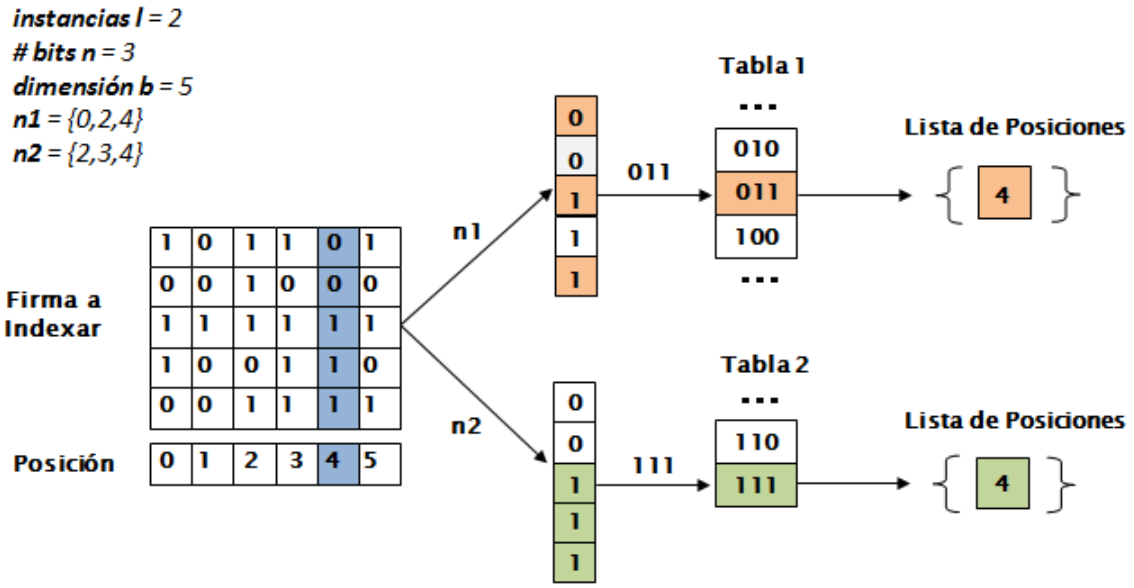


Figura 2.3: Proceso de indexado de la firma de audio.

El principio detrás del método esta basado en que la probabilidad de colisión de dos puntos p y q esta relacionada con la distancia entre ellos. Específicamente, entre más grande es la distancia entre dos objetos menor es la probabilidad de colisión.

Los algoritmos **??** y **??** describen los algoritmos de preprocesamiento y búsqueda para el índice LSH que se utilizó en este trabajo.

En resumen la calidad de los resultados entregados por el índice LSH dependerá de 2 factores principalmente:

- La cantidad de instancias LSH que se utilicen: usar más instancias mejora la calidad de los resultados pero aumenta también los requerimientos de memoria.
- La cantidad de bits n que que se utilicen para crear el hash: un valor de n bajo creará instancias con cubos muy poblados y, por lo tanto, se revisarán muchos candidatos; por otro lado un valor de n alto creará instancias con cubos poco poblados y se revisarán menos candidatos.

Algoritmo 1: Algoritmo de preprocesamiento del índice LSH

Entrada: P,l

Salida : T

```
1 P ← todos los vectores;
2 s ← tamaño de P, size(P);
3 l ← número de tablas hash;
4 for i ← 1 to l do
5 |   Inicializar la tabla hash T_i generando una función hash aleatoria g_i();
6 for i ← 1 to l do
7 |   for j ← 1 to s do
8 |   |   Almacenar cada punto p_j en el cubo g_i(p_j) de la tabla hash T_i
9 Regresar todas las tablas T;
```

La elección de la cantidad de mapas y el número de bits a utilizar dependerá en particular de los requerimientos y recursos de la aplicación para la cual se utilice el índice LSH.

Algoritmo 2: Algoritmo de búsqueda del índice LSH

Entrada: q,T,l

Salida : C

1 $q \leftarrow$ *vector de consulta*;

2 $T \leftarrow$ *conjunto de tablas hash*;

3 $l \leftarrow$ *número de tablas hash*;

4 $C \leftarrow \{\}$; `// Lista de candidatos`

5 ;

6 **for** $i \leftarrow 1$ **to** l **do**

7 $\quad C \leftarrow C \bigcup \{c | c$ *es un candidato encontrado en el cubo* $g_i(q)$ *de la tabla* $T_i\}$;

8 *Regresar el conjunto de candidatos* C;

2.3. Resumen de Capítulo

En este capítulo se presentaron los fundamentos teóricos de funcionamiento de la huella de audio basada en la entropía de la señal y el índice LSH. Una vez que han sido introducidos estos conceptos se puede proceer a la explicación sobre el diseño y la implementación del seguidor de audio. Dicho será el enfoque del próximo capítulo donde se brindará una descripción a detalle.

Capítulo 3

Diseño e Implemetación

En este capítulo se discuten algunas de las decisiones de diseño e implementación del seguidor y se brinda una descripción del funcionamiento general del mismo. Se introducen además un número de parámetros utilizados, se explica su propósito y la relación entre ellos.

3.1. Definición de Conceptos

Para poder tratar con el diseño del seguidor primero es necesario definir algunos términos que serán utilizados frecuentemente a través de este documento:

- **Interpretación:** un intento realizado por un músico específico para tocar una pieza de música en un momento dado.
- **Intérprete:** el músico que toca la pieza musical.
- **Interpretación base o de referencia:** la interpretación que fue procesada de manera $off - line$ y que se utiliza como referencia para alinear a otra interpretación.
- **Interpretación en vivo u** $on - line$**:** la interpretación que se procesa paulatinamente conforme se reciben sus datos y se alinea, usualmente en tiempo real, con la interpretación base.
- **Ritmo:** también llamado *tempo*, es el reloj musical dinámico utilizado por los intérpretes. Usualmente se refiere a la velocidad o la frecuencia con que un intérprete toca las

notas de una interpretación.

- **Partitura:** es la música de forma escrita que un intérprete lee cuando toca una pieza musical.

- **Subfirma:** segmento de la totalidad de la huella de audio de una interpretación. Para este trabajo en particular se refiere a una matriz de [43 X 17] la cual equivale a medio segundo de audio.

Como se menciona en la Sección **??** el seguimiento en la literatura usualmente se refiere al alineamiento en tiempo real entre una interpretación "en vivo" con su correspondiente partitura, representada de alguna manera por alguna representación simbólica (como pueden ser los estados de un Modelo Oculto de Markov, HMM). El término en Inglés *Score Following* podría ser traducido entonces de forma más precisa como "Seguimiento de Partitura".

El presente trabajo adopta un enfoque diferente a la mayoría de los mencionados en la Sección **??**. No se utiliza programación dinámica (DP), funciones de densidad de probabilidad (PDF's), filtro de Kalman, ni HMM's. Adicionalmente, el seguimiento no se realiza nota a nota, es decir, el alineamiento no se efectúa contra la partitura de la interpretación. En su lugar el seguimiento de audio, como se define en este trabajo, básicamente consiste en alinear una **interpretación "en vivo"** con una **interpretación base** que fue procesada (extracción de características e indexamiento) previamente.

Esta diferencia en el enfoque utilizado provoca que el esquema de seguimiento presentado en este trabajo sea difícil de comparar con otros esquemas de seguimiento desarrollados previamente. Al no ser un seguimiento nota a nota, no se cuenta con un porcentaje de falsos positivos o falsos negativos como en otros esquemas. Por la razón anterior fue necesario idear un esquema propio de validación el cual se presenta en un capítulo posterior.

Así pues, el enfoque de este trabajo no consiste el alinear una interpretación con su partitura sino con otra interpretación utilizada como referencia, por lo tanto, no coincide enteramente con el término *score following*. Es por ello que se propone utilizar el término *Audio Following* o *Seguimiento de Audio*. Cuando se utiliza el índice para realizar el

seguimiento se propone el uso del término *Index Audio Following* (*IAF*) o *Seguimiento de Audio mediante Índices*.

3.2. Diseño General

En esta sección se describe la idea general detrás del diseño del seguidor. El esquema de seguimiento propuesto se compone de los siguientes pasos:

- Preprocesamiento
    1. Obtención de la firma de la interpretación de referencia.
    2. Indexamiento de la firma de audio obtenida en el paso anterior.
- Seguimiento
    1. Obtención de una subfirma de la interpretación on-line.
    2. Obtención de posiciones candidato a partir de la búsqueda en el índice de los vectores de la subfirma.
    3. Obtención de la lista de vecinos más cercanos mediante la comparación entre la subfirma de la interpretación online y las subfirmas obtenidas a partir de las posiciones candidato.
    4. Elección de la siguiente posición en el seguimiento utilizando la lista de vecinos más cercanos obtenida en el paso anterior basándose en alguna de las políticas implementadas.
    5. Repetir los pasos 1 - 4 hasta que termine la interpretación on-line.

3.2.1. Parámetros para la obtención de la huella de audio

Como se explicó en la Sección **??**, la huella de audio utilizada entrega como resultado secuencias de ceros y unos correspondientes al aumento o disminución de la entropía en cada una de las 24 bandas críticas de Bark (aunque para este trabajo únicamente se utilizan 17). Así una interpretación será representada por una matriz de ceros y unos cuyas dimensiones dependerán de los parámetros utilizados por la firma.

Para este trabajo se utilizó un tamaño de marco de ≈ 93 ms (4 KB para muestras tomadas a 44100 Hz) con un traslape de ≈ 12 ms (512 bytes). Así una interpretación de 1 minuto (60 segundos) se convierte en una firma de ≈ [5000 X 17].

3.2.2. Indexamiento de la huella

Cuando se ha extraído la huella de audio de la interpretación base se procede a indexarla. Los conceptos generales de funcionamiento del índice LSH utilizado se describen en la Sección **??**. En la Figura **??** se presentó un ejemplo del mismo. En resumen:

- Se determina cuántos mapas hash (instancias, l), se utilizarán.
- Se determina cuántos bits (n) se utilizarán para crear los hashes.
- Para cada instancia se determinan al azar las posiciones de los bits que serán utilizados para crear el hash.
- Para cada vector binario de la huella se calcula la función hash para cada instancia y se almacena la posición del vector en el cubo adecuado.

En la tabla **??** se observa un ejemplo de la función utilizando $l = 2$ y $n = 3$ para un vector muestra.

Vector Binario	1 0 1 0 0 1 1 0 0 1 1 1 0 1 1 0 0	
Instancia	Posiciones	Hash(Cubo)
1	3,7,10	001
2	0,9,16	110

Tabla 3.1: Ejemplo función hash.

3.2.3. Obtención de posiciones candidato

Una vez que la huella de audio de la interpretación base se ha indexado completamente se ha terminado con el preprocesamiento y se puede comenzar a realizar el seguimiento. Para realizar el alineamiento se utilizarán segmentos de 0.5 segundos de audio de la interpretación on-line de los cuales se calcula la huella. Una vez que se tiene una subfirma equivalente a 0.5 segundos de audio (≈ [43 X 17]) el siguiente paso será determinar la

posición donde correspondería dicha subfirma en la firma de la interpretación base. Para ello se buscará en el índice las posibles posiciones donde podría ocurrir dicha correspondencia. A estas posiciones se les llama posiciones candidato.

En el Algoritmo **??** se describe en forma algoritmica el procedimiento para la obtención de las posiciones candidato. Se inicia con una lista de candidatos vacía. Por cada vector en la subfirma se crean variaciones de 1, 2 o hasta 3 bits como se menciona en (**?**), (**?**) para aumentar la robustez de la búsqueda. Por cada vector de variación creado se calcula su función hash para cada instancia del índice y se obtienen las posiciones almacenadas en los cubos correspondientes. Estas posiciones son agregadas a la lista de candidatos si es que no se encuentran ya en la lista.

El crear variaciones de 1, 2 o 3 bits de cada vector de la subfirma tiene consecuencias importantes en el funcionamiento del seguidor. Utilizar variaciones de 3 bits generará una cantidad de candidatos muy grande lo cual puede afectar de manera significativa la cantidad de comparaciones entre subfirmas que se efectúan, como se describe en la sección posterior y, por lo tanto, en el tiempo de ejecución del algoritmo.

Cuando ha terminado este paso se cuenta con todas las posibles posiciones consideradas como candidatos para la alineación.

3.2.4. Obtención de knn

El siguiente paso en el seguimiento consiste en producir una lista de vecinos más cercanos de entre todas las posiciones candidato que se obtuvieron en el paso anterior. Para ello se comparará la subfirma de la interpretación on-line con cada una de las subfirmas de la interpretación base conseguidas a partir de las posiciones candidato.

Para la comparación de firmas se utiliza una de tres medidas de distancia implementadas, las cuales se describen en una sección posterior. Es altamente improbable que se encuentre una subfirma dentro de la interpretación base que sea exáctamente igual ($distancia = 0$) a la subfirma que se busca de la interpretación on-line debido a que se trata de 2 interpretaciones distintas. Incluso sería difícil si se tratara de la misma interpretación simplemente degradada.

Por la razón anterior, y para aumentar la robustez del seguimiento, no se considera

Algoritmo 3: Algoritmo para la obtención de todos los posibles candidatos.

```
Entrada: mtlf,T
Salida  : candidatos
1  mtlf ← La matriz de medio segundo de audio.;
2  T ← conjunto de tablas hash.;
3  candidatos ← {}          // Lista de posibles candidatos.;
4  for v ∈ mtlf do                    // Por cada vector en la matriz.
5      vars ← crearVariaciones(v)  // variaciones de 1, 2 o 3 bits del
       vector, sección ??;
6      for v ∈ vars do                          // Por cada variación.
7          for t ∈ T do                             // Por cada tabla.
8              bits ← bits utilizados por la instancia.;
9              h ← hash(v, bits)          // Se obtiene el hash.;
10             bucket ← t.obtener(h)      // Se obtiene el bucket del hash
               calculado.;
11             for posicion ∈ bucket do
12                 if !candidatos.contiene(posicion) then
13                     candidatos.agregar(posicion);
```

únicamente la posición cuya subfirma entregue la distancia más pequeña respecto a la subfirma buscada. En cambio se obtiene una lista de k vecinos más cercanos knn integrada por las k posiciones que tengan las distancias más pequeñas a la subfirma buscada. Alguna de estas posiciones será elegida como la siguiente posición en el siguiente paso.

El Algoritmo **??** describe el proceso para obtener dicha lista. Se entregarán dos listas al terminar el algoritmo: la lista knn que contendrá las posiciones y la lista $distanciasKnn$ que contendrá las distancias que reportan las subfirmas de las posiciones correspondientes contenidas en knn. Para cada candidato en la lista de posibles candidatos, si no se ha revisado anteriormente, se obtiene su subfirma. Posteriormente, esta subfirma es comparada con aquella de la interpretación on-line usando alguna de las distancias implementadas.

Si cumple los requisitos, la posición candidato y su correspondiente distancia reportada se incluyen en las listas *knn* y *distanciasKnn*. Lo anterior se repite hasta que se han revisado todos los candidatos.

Se ha dejado intencionalmente vaga la manera en que se obtienen los k vecinos más cercanos puesto que el propósito de esta sección no es brindar una explicación sobre los algoritmos pertinentes. Basta con saber que en las listas de *knn* y *distanciasKnn* se tienen las posiciones adecuadas y sus respectivas distancias.

Algoritmo 4: Algoritmo para la obtención de la lista de vecinos más cercanos.

```
Entrada: mtlf,candidatos
Salida   : knn, distanciasKnn
1  mtlf ← matriz de medio segundo de audio.;
2  candidatos ← lista de posiciónes candidato.;
3  indicesRevisados ← {}     // Lista de candidatos ya revisados.;
4  knn ← {}.                 // Lista de vecinos más cercanos.;
5  distanciasKnn ← {}        // Distancias respectivas a los vecinos más
   cercanos.;
6  for c ∈ candidatos do                          // Por cada candidato.
7      if !indicesRevisados.contiene(c) then
8          indicesRevisados.agregar(c);
9          matrizCandidato ← obtenerSubfirma(c)   // Se obtiene una
           matriz de 0.5 segundo a partir de la posición candidato;
10         distancia ← distanciaEntreMatrices(mtlf, matrizCandidato);
11         if !indicesRevisados.contiene(c) then
12             knn.agrega(c);
13             distanciasKnn.agrega(distancia);
```

Es importante mencionar que al obtener la subfirma de la posición candidato (método *obtenerSubfirma*) se toma en cuenta la posición dentro de la matriz que tenía el vector a partir del cual se obtuvo la posición candidato. Por ejemplo, si se obtuvo la

posición candidato $cp = 5$ a partir del segundo vector de la subfirma a alinear, al obtener la subfirma de la interpretación base se utilizará la posición 5 como la segunda posición, es decir, la subfirma se obtendrá a partir de la posición 4. Lo anterior para aumentar la probabilidad de que las subfirmas sean lo más parecidas posible.

En la Figura **??** se ejemplifica el procedimiento completo de búsqueda de candidatos en el índice descrito en las secciones previas. Se tiene la subfirma de la interpretación on-line llamada "Subfirma a buscar" en la figura. Se toma cada uno de sus vectores, para el ejemplo se tomó el vector de la posición 4. Se crean variaciones de 1, 2 o 3 bits, en el ejemplo se crearon variaciones de 1 bit únicamente. Se toma cada variación y se calcula su hash para cada tabla. En el ejemplo se utiliza una instancia donde se utilizan los bits 0, 2 y 4 para calcular la función hash. Una vez que se obtiene el hash del vector se busca en el cubo correspondiente ($hash = 011$) de la tabla hash y se revisan las posiciones que contiene. En el ejemplo la primera posición del cubo es la posición 25, entonces se debería obtener una subfirma de la interpretación base que comience en el índice 25. Sin embargo, como el vector a partir del cual se crearon las variaciones se encontraba en la posición 4 de la subfirma on-line entonces existe un defasamiento (*offset*) de 4 posiciones. Por lo tanto la subfirma se obtiene a partir de la posición 21 ($25 - 24 = 21$) de la huella de referencia.

3.2.5. Elección de la siguiente posición

El último paso en el algoritmo de seguimiento consiste en la elección de la que será considerada como la siguiente posición. Como se mencionó en la sección anterior, una de las posiciones incluidas en la lista de k vecinos será elegida como posición siguiente, a menos que el algoritmo decida no avanzar y permanecer en la posición anterior.

Inicialmente se asume la posición inicial $sp = 0 = pp$. A partir de ahí, debido a la precedencia temporal de los eventos, se elegirá siempre una posición siguiente que sea superior o igual a la posición anterior $np \geq pp$. Es decir, asumimos que el intérprete no "regresará" dentro de la interpretación sino que siempre "avanza" o se "mantiene" en una posición. La elección de la siguiente posición está sujeta a ciertas politicas que se describen posteriormente de manera extensa.

El proceso se repite hasta que la interpretación on-line termina y se han alineado

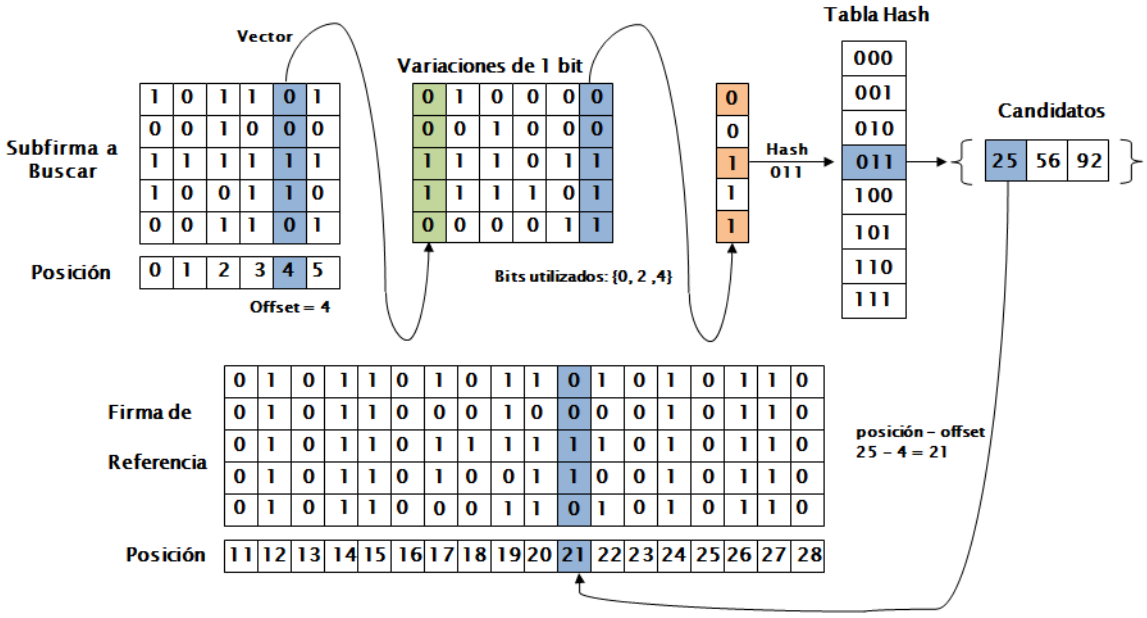


Figura 3.1: Proceso de obtención de un candidato.

todas sus subfirmas con la interpretación base con lo cual se da por terminado el seguimiento.

3.3. Distancias

En esta sección se describen los 3 tipos de distancias implementadas para la comparación de subfirmas que se necesita durante el seguimiento.

3.3.1. Distancia de Hamming

La distancia de Hamming es una distancia para comparación de cadenas introducida por Richard Hamming en (**?**). Consiste simplemente en comparar caracter por caracter ambas cadenas y registrar en cuantas posiciones dichos caracteres son diferentes. Por ejemplo:

1. La distancia de Hamming entre 10011 y 10101 es de 2.

2. La distancia de Hamming entre 00101 y 10000 es de 3.

3. La distancia de Hamming entre 11001 y 11000 es de 1.

Las ventajas de la distancia de Hamming es que es sumamente sencilla de implementar y muy rápida de evaluar. Entre sus desventajas se encuentra la necesidad de que ambas cadenas sean de la misma longitud. Para comparar cadenas de diferentes longitudes, donde no sólo se consideran substituciones sino también inserciones y borrados, se necesita una métrica más robusta como la distancia de Levenshtein.

3.3.2. Distancia de Levenshtein

La distancia de Levenshtein o distancia de edición, es una distancia para la comparación de cadenas introducida por Vladimir Levenshtein en (**?**). Informalmente, consiste en el número mínimo de ediciones de un caracter (substitución, inserción, borrado) requeridas para convertir una cadena en otra. Por ejemplo:

1. La distancia de Levenshtein entre 111 y 10101 es de 2 (2 borrados).

2. La distancia de Levenshtein entre 1001 y 1000 es de 1 (1 substitución).

3. La distancia de Levenshtein entre 001 y 01 es de 1 (1 inserción).

La distancia de Levenshtein es ampliamente utilizada en diversas aplicaciones de comparación de cadenas como los correctores de ortografía. Sin embargo, la mayor desventaja de esta distancia es su costo de cálculo, el cual es aproximadamente proporcional al producto de la longitud de ambas cadenas.

3.3.3. Subsecuencia Común Más Larga, LCS

Esta distancia consiste en encontrar la subsecuencia común más larga entre 2 o más sequencias de caracteres. Se denomina LCS por sus siglas en Inglés (*Longest Common Subsequence*). Para un número constante de secuencias (2 para el presente trabajo) el problema es soluble en tiempo polinomial utilizando programación dinámica. Ejemplos:

1. La LCS entre 10011 y 10101 es 1001.

2. La LCS entre 001010 y 10000 es 0000.

3. La LCS entre 101 y 010 es 10.

Esta métrica únicamente permite inserciones y borrados, no incluye sustituciones. Para convertir la métrica LCS a una distancia se propone la diferencia de longitud entre la LCS obtenida y la longitud de la secuencia más larga:

$$distance = max(length(seq1), length(seq2)) - length(lcs)$$

Así para los ejemplos presentados las distancias serían: 1, 2 y 1 respectivamente.

3.3.4. Comparación entre las distancias

Como es de esperarse, se obtienen valores de distancia diferentes al utilizar las diversas distancias implementadas como se muestra en el ejemplo siguiente:

1. La distancia de Hamming entre 11010101110 y 01101000110 es 6.

2. La distancia de Levenshtein entre 11010101110 y 01101000110 es 3.

3. La distancia LCS entre 11010101110 y 01101000110 es 2 (LCS = '110100110').

Estas diferencias pueden tener un impacto muy significativo en el comportamiento del seguimiento. Lo anterior debido a que la lista de k vecinos más cercanos obtenida puede ser sumamente distinta dependiendo de la distancia de comparación utilizada. Las comparaciones del comportamiento del seguimiento con las diversas distancias se discutirán en el Capítulo **??**.

3.4. Modos de Seguimiento

En esta sección se presentan las cuatro formas diferentes que se implementaron para elegir la siguiente posición a partir de la lista de k vecinos más cercanos. A estas formas se les denominó "modos":

- Modo 1: índice más cercano en tiempo.

- Modo 2: índice más cercano en distancia.

- Modo 3: combinación de los Modos 1 y 2.
- Modo 4: índice más cercano a la predicción.

3.4.1. Modo 1: índice más cercano en tiempo

La manera más sencilla de elegir la siguiente posición en el seguimiento consiste en simplemente elegir el índice más cercano (mayor) a la posición previa. De la lista de k vecinos más cercanos entregada por la búsqueda en el índice se descartan aquellos que sean menores o iguales a la última posición conocida. Se obtiene la diferencia de los restantes respecto a la posición previa y se elige como siguiente posición aquel que tenga la menor diferencia.

Por ejemplo, si nuestra posición previa es $pp = 100$ y la lista de knn entregada es la mostrada en la Tabla **??**. Para este modo se elegiría el índice 105 como siguiente posición debido a que es mayor a 100 y tiene la menor diferencia (5). El índice 97 sería descartado ya que es menor a 100.

índice: 110	distancia: 430
índice: 120	distancia: 432
índice: 97	distancia: 435
índice: 105	distancia: 520
índice: 200	distancia: 521

Tabla 3.2: Ejemplo de lista de knn

Se puede apreciar los problemas que podría presentar el Modo 1. Para que el seguimiento sea lo más adecuado posible, sería deseable que la siguiente posición se eligiera entre aquellas que se encuentran más cercanas en distancia al fragmento de audio que estamos buscando. Es claro que el índice 110 o incluso 120 serían mejores candidatos para ser designados como siguiente posición puesto que reportan valores de distancia mucho menores. Sin embargo, la heurística del modo 1 los descarta y en su lugar elige el índice 105, el cual tiene una distancia muy superior a los índices 110 o 120.

Este modo favorece los "pasos pequeños" lo cual podría ser no deseable en algunos casos ya que el seguimiento podría quedarse "estancado" y no avanzar de la manera

adecuada.

3.4.2. Modo 2: índice más cercano en distancia.

Una segunda política para la elección de la siguiente posición consiste en elegir el índice dentro de la lista de vecinos más cercanos que tenga la menor distancia al segmento de audio que estamos buscando (cumpliendo que debe ser una posición "superior" a la anterior).

Para el mismo ejemplo utilizado en la Sección **??** el índice elegido sería el 110 puesto que es superior a la posición anterior (100) y es el que reporta la menor distancia.

Para ésta política también existen problemas potenciales. Por ejemplo, debido a la naturaleza cíclica de la música es altamente probable que varias secciones de una interpretación/canción se parezcan mucho entre sí. Los coros de las canciones son un muy buen ejemplo. De esta manera, el índice elegido por la política del modo 2 podría ser un índice que se encuentre sumamente lejano a la posición anterior lo cual podría ser no deseable en ciertas circunstancias.

Tome el siguiente ejemplo, si nuestra posición previa es $pp = 100$ y la lista de *knn* entregada es la mostrada en la Tabla **??**.

índice: 300	distancia: 101
índice: 120	distancia: 103
índice: 480	distancia: 104
índice: 99	distancia: 110
índice: 101	distancia: 112

Tabla 3.3: Ejemplo 2 de lista de *knn*

El índice elegido como siguiente posición sería el índice 300 ya que reporta una distancia de 101, sin embargo nuestra posición anterior era el índice 100 por lo cual existe una diferencia de 200 posiciones entre la posición anterior y la siguiente. Ésto podría ser deseable, por ejemplo, si justo en esa posición la interpretación que se está siguiendo tiene una parte faltante respecto a la interpretación base que se encuentra indexada. Como si el entérprete se "saltara" algunas notas.

Pero para el ejemplo presente, es claro que la siguiente posición debería ser el

índice 120 que se encuentra a sólo 20 posiciones de la posición anterior y la distancia entre las huellas es únicamente de 2 puntos más que el índice 300. Este problema se mitiga un poco gracias al parámetro $windowMultiplier$ el cual se aborda en una sección posterior.

3.4.3. Modo 3: combinación de los Modos 1 y 2.

Es claro a partir de las secciones anteriores que sería deseable encontrar un equilibrio entre elegir el índice más cercano a la posición anterior en tiempo o el más cercano en distancia. La política del modo 3 pretende resolver este problema.

Llamemos *pp* a la posición previa, se calcula la siguiente posición en tiempo *npt*(sección **??**) y en distancia *npd* (sección **??**). Así como sus respectivas distancias *dt* y *dd*. Como el Modo 1 se basa en la cercanía en tiempo, es posible que la distancia reportada por la subfirma de la posición obtenida con este modo sea mucho mayor a la distancia reportada por la posición del Modo 2. Para lidiar con ésto, se utilizará un valor umbral que sirve para distinguir si la posición reportada por el Modo 1 tiene una distancia mucho mayor a la reportada por el Modo 2. Se aplican siguientes criterios:

- Si la diferencia entre *dt* y *dd* es mayor al umbral ($dt - dd > umbral$) asignamos *npd* como siguiente posición.

- Si la diferencia entre *dt* y *dd* es menor o igual al umbral ($dt - dd \leq umbral$) significa que ambas posiciones deberán ser tomadas en cuenta:

  - Si *npt* se encuentra "mucho más cercana" a *pp* que *npd* ($(npt - pp) \ll (npd - pp)$) entonces se elige como siguiente posición *npt*.

  - De lo contrario, se elige como siguiente posición *npd*.

Esta política combina las dos anteriores pues se favorece la elección del candidato más parecido siempre y cuando no se encuentre muy lejano de la posición anterior.

Lo anterior se puede apreciar en forma algorítmica en el Algoritmo **??**:

Algoritmo 5: Algoritmo para la elección de la mejor posición entre tiempo y distancia.

Entrada: *pp*,*npt*,*npd*,*dt*,*dd*,*umbral*

Salida : *np*

```
1  pp ← Posición previa.;
2  npt ← Posición siguiente elegida por cercanía en tiempo.;
3  npd ← Posición siguiente elegida por cercanía en distancia.;
4  dt ← Distancia reportada por la subfirma de npt.;
5  dd ← Distancia reportada por la subfirma de npd.;
6  (umbral ← Valor umbral definido previamente.);
7  np ← pp.                    // Posición siguiente.;
8  if dt − dd > umbral then
9      np ← npd;
10 else
11     if (npt − pp) ≪ (npd − pp) then
12         np ← npt;
13     else
14         np ← npd;
```

3.4.4. Modo 4: índice más cercano a la predicción

Se implementó una cuarta política que se basa en la idea de que los cambios muy brucos son poco comunes, salvo en casos extremos, y que se puede realizar una "predicción" de cuál será la siguiente posición a partir de las posiciones anteriores.

Para ésto se almacena no sólamente la posición anterior *pp* sino también la anterior a la anterior *ppp* y se realiza una extrapolación lineal a partir de ambas para obtener la predicción de la siguiente posición *npp*. Posteriormente se elige el índice dentro de la lista de vecinos más cercanos que se encuentre más cerca *npp*. Por ejemplo, si tomamos la lista de vecinos mostrada en la Tabla **??** y $npp = 110$ se elegiría como siguiente posición el índice 101 puesto que $abs(101 - 110) = 9$. Si $npp = 111$ entonces se elegiría el índice 120 puesto

que $abs(120 - 111) = 9$.

3.5. Parámetros

En esta sección se describen los parámetros importantes utilizados en la presente implementación de seguimiento de audio, así como su papel e impacto en el comportamiento del mismo.

La sección se divide en 2 subsecciones: una de ellas dedicada a los parámetros que afectan el algoritmo de seguimiento en general y otra dedicada a los parámetros que afectan al índice LSH.

3.5.1. Parámetros Generales

Estos parámetros afectan el funcionamiento del algoritmo de seguimiento sin importar si la búsqueda de posibles candidatos se realiza de forma secuencial o utilizando el índice LSH.

Parámetro k: El parámetro k se refiere al tamaño de la lista de vecinos más cercanos que se utilizará para elegir de entre ellos a la siguiente posición, tiene un impacto importante en el seguimiento puesto que limita las posibles opciones a elegir:

- Si se elige un valor k muy pequeño se corre el riesgo de dejar fuera posibles candidatos viables. Podría ser que la lista se llene con candidatos que no son viables por diversas razones: se encuentran "atrás" o demasiado alejados de la posición previa y el algoritmo podría optar por "mantenerse" en la posición previa cuando debería de "avanzar". Una ventaja de tener un valor pequeño es que la búsqueda tarda un poco menos.

- Si se elige un valor de k muy grande se corre el riesgo de incluir entre los posibles candidatos a posiciones cuyas subfirmas en realidad no son similares a la subfirma buscada. El algoritmo podría decidir "avanzar" cuando en realidad debería "mantenerse". Una ventaja de aumentar el valor de k es que aumenta la posibilidad de encontrar una posición que cumpla las condiciones para ser elegida como posición siguiente.

El parámetro k, como muchos otros, se puede optimizar para cada caso particular de manera que el seguimiento se realice de la mejor manera posible. Sin embargo, gracias a los experimentos realizados (algunos de los cuales se pueden encontrar en el Capítulo **??**) se propone utilizar un valor de k entre 10 y 50.

Parámetro nB: Este parámetro se refiere a cuántas de las 24 bandas críticas de Bark que contiene la huella de audio se utilizarán para realizar las comparaciones entre matrices. Ésto toma particular importancia cuando las canciones o interpretaciones utilizadas en el seguimiento no contienen componentes de frecuencia en todas las bandas críticas.

- Si se utiliza un número muy grande de bandas críticas (mayor a la cantidad de bandas con contenido útil) se puede afectar el resultado de la comparación entre las matrices. En particular al utilizar las distancias de Levenshtein y LCS las cuales toman en cuenta las inserciones y borrados. Adicionalmente, el tiempo de comparación aumenta puesto que las matrices son de mayor tamaño.

- Igualmente si se utiliza un número demasiado pequeño de bandas críticas, se corre el riesgo de no tomar en cuenta informacion importante que podría ser determinante en la elección correcta de la posición siguiente.

Lo recomendable entonces sería utilizar el número mínimo de bandas críticas de Bark que alcancen a cubrir el espectro de frecuencia de las interpretaciones utilizadas.

Como se describe también en el Capítulo **??**, para los experimentos de este trabajo se utilizaron interpretaciones de piezas clásicas tocadas en piano. Un piano de 88 teclas contiene 8 octavas por lo que su rango de frecuencia va de los 27.5Hz (A0) hasta los 4186.01Hz (C8). Debido a ésto, se utilizaron únicamente las primeras 17 bandas críticas de Bark las cuales comprenden un rango de 20Hz - 3700 Hz con lo cual se cubre casi a cabalidad el espectro de frecuencia utilizado. Quedando fuera únicamente las últimas 2 teclas del piano B7(3951.07Hz) y C8(4186.01Hz).

Parámetro windowSize: El parámetro *windowSize* se refiere al tamaño que tendrá la subfirma a buscar. Como se mencionó en la Sección **??** se usarán subfirmas que

equivalen a medio segundo de audio por lo cual para marcos con un traslape de 11.6 ms este parámetro tendrá un valor de $windowSize = 43$ y las subfirmas serán de dimensiones [43 X 17]

Si se varía el parámetro *windowSize* simplemente se cambia el tiempo al cual equivale la subfirma a buscar. Por ejemplo si el valor se reduce a 22, se tratará con subfirmas equivalentes a 0.25s; si se aumenta el valor a 86 se tratará con subfirmas equivalentes a 1s.

Estas variaciones tienen sus respectivas ventajas y desventajas:

- Si se trata con subfirmas de 0.25 s, el seguimiento podrá ser mucho más fino, en el sentido de que se podrá tener un alineamiento cada cuarto de segundo. Sin embargo, un cuarto de segundo podría ser tiempo insuficiente para realizar todo el procesamiento necesario para realizar el alineamiento: es necesario capturar el audio, obtener la huella, una vez que se tiene la subfirma completa se buscan los posibles candidatos, se obtiene la lista de k vecinos y se elige la siguiente posición de acuerdo a la política activa (véase Sección **??**). Para las aplicaciones en tiempo real, todo lo anterior se realiza mientras paralelamente se continúa capturando el audio para la siguiente subfirma a buscar. Si se utiliza una distancia como Levenshtein la cual es costosa de calcular, las comparaciones entre subfirmas requieren mucho más tiempo y podría ser que no se cumplan los requerimientos de tiempo real.

- Si se trata con subfirmas mayores a 1 segundo, el tiempo puede dejar de ser problema. Sin embargo el seguimiento ya no sería tan preciso lo cual puede ser desastrozo para ciertas aplicaciones.

Por las razones expuestas en los párrafos anteriores y por los resultados obtenidos (véase el Capítulo **??**) en el presente trabajo se utilizan subfirmas de medio segundo de audio ($windowSize = 43$).

Parámetro windowMultiplier: Este parámetro se utiliza como una manera de enforzar la política que dice que la siguiente posición no puede estar muy alejada de la posición anterior. Es decir, limita los posibles "saltos" que puede dar el seguimiento.

Se utiliza en conjunto con el parámetro *windowSize* de la siguiente manera: se

establece un límite inferior para el parámetro, por ejemplo $windowMultiplierDefault = 3$. Si $windowSize = 43$ (0.5 seg.) como se describió en la sección anterior, eso implica que la posición siguiente no podrá estar a más de 129 posiciones de distancia de la posición anterior ($43*3 = 129$). En otras palabras, no podrá estar a más de 1.5 segundos. Si la posición elegida de entre los posibles candidatos no cumple con ésta condición (por ejemplo: $pp = 100$ y $pnp = 300$) entonces el algoritmo de seguimiento determinará que es más conveniente mantenerse en la posición actual $np = pp$.

Lo anterior puede ocasionar comportamientos no deseados: conforme avanza el audio de la interpretación "online" es más probable que si no se avanzó en el paso anterior los candidatos a siguiente posición se encuentren más alejados de pp lo cual haría que la condición

$$np \leq pp + (windowSize * windowMultiplier) \tag{3.1}$$

no se cumpla jamás.

En el Algoritmo **??** se muestra la solución propuesta para dicha situación. Si se cumple la condición de tolerancia **??**, se asigna como siguiente posición la posición elegida por la política activa. El parámetro $windowMultiplier$ se reduce siempre y cuando esté por encima del mínimo $windowMultiplierDefault$. Si no se cumple la condición de tolerancia, entonces se asigna como siguiente posición la posición previa (no se avanza) y se aumenta el valor de $windowMultiplier$ para aumentar el rango de posiciónes permitido en la siguiente iteración.

Algoritmo 6: Algoritmo de modificación del parámetro $windowMultiplier$

```
1 if pnp ≤ pp + (windowSize * windowMultiplier) then
2     np ← pnp;
3     if windowMultiplier > windowMultiplierDefault then
4         windowMultiplier − −;
5 else
6     np ← pp;
7     windowMultiplier + +;
```

En resumen, si se avanza en el seguimiento se decrementa el valor de $windowMultiplier$ siempre y cuando sea mayor al valor mínimo. Si no se avanza en el seguimiento, se aumenta el valor de $windowMultiplier$ con el fin de permitir que el seguimiento se reponga del posible retraso.

El límite inferior del parámetro $windowSize$ llamado $windowSizeDefault$ o wmd indica cuál será la dimensión esperada de los posibles "saltos" en la interpretación. Si se le asigna un valor muy grande podría ocasionar que el seguimiento no sea adecuado gracias a la posibilidad de elegir posiciones muy lejanas a la posición previa dependiendo de la política de selección utilizada (véase la sección **??**).

Parámetro distance: El parámetro *distance* simplemente indica cuál de las 3 distancias descritas en la sección **??** se utiliza para realizar la comparación de las matrices.

1. Distancia de Levenshtein.
2. Distancia de Hamming.
3. Distancia LCS.

Parámetro mode: El parámetro *mode* únicamente indica cuál de las 4 políticas descritas en la sección **??** se utiliza para realizar la elección de la posición siguiente.

1. Índice más cercano en tiempo.
2. Índice más cercano en distancia.
3. Índice más adecuado entre tiempo y distancia.
4. Índice más cercano a la predicción.

3.5.2. Parámetros LSH

Estos parámetros únicamente afectan el funcionamiento del índice LSH por lo cual no modifican directamente el comportamiento del algoritmo de seguimiento. Sin embargo, si pueden tener un impacto importante puesto que determinan cuáles serán los índices a ser revisados para posteriormente poder ser considerados como posibles candidatos.

Los primeros dos parámetros que se describirán enseguida, $howManyMaps$ y $howManyBits$, determinan la manera en que se creará el índice LSH. El parámetro $bitVariations$ se utiliza cuando se va a realizar una búsqueda en el índice.

Parámetro howManyMaps: El parámetro $howManyMaps$ determina cuantas instancias utilizará el índice LSH. Aumentar o disminuir este parámetro tiene consecuencias bastante obvias:

- Si se aumenta el número de instancias utilizadas, aumentará la respuesta del índice puesto que se obtendrán más cubos por cada vector binario analizado (1 cubo por cada mapa). Aumentará el tiempo de procesamiento por la misma razón.
- Si se disminuye el número de mapas el efecto es contario, disminuye el tiempo de procesamiento pero también disminuye la cantidad de posibles posiciones candidato.

Parámetro howManyBits: El parámetro $howManyBits$ determina cuántos bits de los 17 disponibles se utilizarán para obtener la función hash de los vectores binarios. Este parámetro es sumamente importante puesto que determina que tan "densamente poblados" estarán los cubos de cada instancia.

Para ejemplificar lo anterior recordemos la siguiente situación: una interpretación de 1 minuto se convierte en aproximadamente 5000 vectores binarios (considerando marcos con un traslape de 11.6 ms). Consideremos siguientes 2 ejemplos:

1. Si se utilizan 14 bits, existirán 16,384 posibles cubos. Si los hashes se distribuyeran uniformemente significaría que aproximadamente 11,000 de los 16,000 cubos estarían vacíos y 5,000 tendrían un elemento únicamente. Como cada vector de la firma produce un hash único por cada instancia, ésto implicaría que por cada vector únicamente se revisarían $1 * howManyMaps$ posiciones lo cual podría ser insuficiente para tener una respuesta adecuada.
2. Si se utilizan 6 bits, existirán únicamente 64 posibles cubos en cada tabla. Si suponemos de nuevo que los hashes se distribuyen uniformemente cada cubo tendría ≈ 78 elementos. Lo cual implica que por cada vector se revisarían $78 * howManyMaps$ lo

cual podría ocasionar que el índice haga más comparaciones que usando búsqueda secuencial (dependiendo de la cantidad de instancias y la longitud de la interpretación).

Adicionalmente, debido a que el conjunto de bits a utilizar por cada instancia se elige aleatoriamente con cada ejecución, 2 ejecuciones distintas con los mismos parámetros pueden obtener resultados distintos.

Parámetro bitVariations: El parámetro $bitVariations$ determina cuantas variaciones de cada vector binario se crearán con el fin de aumentar el número de cubos revisados por instancia. Las posibles opciones son 1, 2 y 3 las cuales indican si se crearán todas las posibles variaciones de 1, 2 o 3 bits respectivamente.

Este parámetro se incluyó con el fin de compensar que es poco probable que se encuentre un vector binario en la huella de la interpretación on-line tal cual como se encuentra en la huella de la interpretación base debido al posible ruido o cualquier otro tipo de degradación. Así, si un vector de la interpretación base es similar más no igual a otro de la interpretación on-line (tienen 1, 2 o 3 bits de diferencia), también se considerará dicho vector como posible candidato.

Por ejemplo, si se crean todas las posibles variaciones de 1 bit para un vector binario de 17 bits se generarán 17 vectores alternativos. Si usamos 6 bits para generar el hash de un vector, 11 de los 17 vectores alternativos generarán el mismo hash que el vector original. Sin embargo, 6 de ellos generarán un hash diferente, lo cual ocaciona que se revisen 6 cubos adicionales por cada instancia. Siguiendo con el ejemplo de la sección anterior, el número total de candidatos a revisar sería $78 * howManyMaps * 6$.

Consecuentemente, crear todas las posibles variaciones de 2 o 3 bits aumentará exponencialmente la cantidad de buckets obtenidos y, por lo tanto, la cantidad de candidatos revisados.

3.5.3. Interacción entre Parámetros

Debería resultar obvio entonces que todos los parámetros mencionados en las 2 secciones anteriores interactúan entre sí, ocasionando cambios en el comportamiento del

algoritmo de seguimiento. También resulta natural el buscar optimizar dichos parámetros con el fin de que el seguimiento se comporte de la manera más adecuada posible.

Algunas de estas interacciones son muy claras, por ejemplo: si aumentar el número de instancias $howManyMaps$ aumenta la cantidad de cubos obtenidos, entonces se puede aumentar también el número de bits que se utiliza para calcular el hash $howManyBits$. Ésto disminuye la "densidad poblacional" de cada cubo con lo cual se mantiene el equilibrio en la cantidad de candidatos finales revisados.

En realidad los 3 parámetros LSH ($howManyMaps$, $howManyBits$ y $bitVariations$) deben de calibrarse de tal manera que se visite la cantidad suficiente de candidatos para obtener un comportamiento equivalente al de la búsqueda secuencial pero haciendo un número considerablemente menor de comparaciones, de manera que se justifique la utilización del índice.

Como nota adicional sobre estos 3 parámetros, es importante mencionar también que en este trabajo se están combinando 2 técnicas diferentes para lidiar con la proximidad: el índice LSH (**?**) y las variaciones de bits (**?**). Anteriormente, se han utilizado por separado. Sin embargo, nunca habían sido utilizadas en conjunto por lo que el presente trabajo es el primero en proponer este esquema.

Otras interacciones podrían no ser tan obvias, por ejemplo: si se permite la posibilidad de hacer saltos grandes con $windowMultiplierDefault$, entonces se debe de reducir la cantidad de vecinos k ya que si no, se corre el riesgo de saltar a una posición muy lejana que no es tan parecida a la matriz que se busca.

En la Tabla **??** del Apéndice **??** se presenta un resumen del efecto de aumentar o disminuir cada parámetro y, en caso de ser necesario, la manera de equilibrar dicho efecto cambiando otro.

3.6. Resumen de Capítulo

En este capítulo se presentó una descripción de la idea propuesta en este proyecto para realizar el seguimiento de audio. Se inició con la definición de algunos términos comunes y posteriormente se expuso a detalle cada uno de los pasos propuestos en conjunto con

sus respectivos algoritmos y ejemplos ilustrativos de funcionamiento. Finalmente, se definieron los parámetros utilizados y la manera en que afectan el comportamiento del seguidor. Habiendo discutido los detalles de diseño e implementación del seguidor, en el siguiente capítulo se presentan algunos de los resultados de los experimentos realizados así como los valores recomendados para los parámetros presentados durante este capítulo.

Capítulo 4

Experimentos y Resultados

En este capítulo se presentan los resultados obtenidos a partir de los experimentos realizados. Para realizar dichos experimentos se utilizaron las grabaciones utilizadas en (**?**) y (**?**). Estas grabaciones consisten de 22 pianistas distintos tocando 2 extractos de piezas en piano de Frédéric Chopin (Etude en E Major, Op. 10, no. 3, barras 1-21; y Ballade Op. 38, barras 1-45, Stereo, 44.1kHz). Los pianistas eran estudiantes o profesores en la Universidad de Música de Vienna. Las partituras de ambas interpretaciones se pueden encontrar en el Apéndice **??**. Todas las grabaciones se pueden obtener en línea en (**?**).

Las interpretaciones del Etude tienen una duración de entre 70.1 a 94.4 segundos, lo cual se transforma en huellas de audio que contienen de entre 6038 a 8130 líneas aproximadamente; así mismo, las interpretaciones de la Ballade tienen una duración de entre 112.2 y 151.5 segundos, sus huellas contienen entre 9664 y 13049 líneas. Se puede apreciar que existen diferencias significativas entre las interpretaciones aún cuando es la misma pieza musical interpretada por pianistas profesionales.

4.1. Validación

Para poder validar el esquema de alineamiento propuesto es necesario efectuar una evaluación del mismo. No obstante, ésta es una tarea compleja debido a que no se cuenta con un alineamiento de referencia base contra el cuál comparar. Se podría intentar realizar un alineamiento manual escuchando secciones de audio de una interpretación y

tratar de encontrar la sección correspondiente en alguna otra interpretación. Sin embargo, este método tomaría demasiado tiempo y sería altamente impreciso.

Por las razones anteriores, en este trabajo se propone la creación de huellas de audio sintéticas, es decir, huellas de audio que han sido modificadas aleatoriamente para simular las posibles diferencias que podrían ocurrir entre diversas interpretaciones. De esta manera se puede evaluar si el resultado entregado por el seguimento es el resultado "esperado" puesto que se conoce de antemano cómo y dónde fue modificada la huella de audio.

Una nota importante es que para los resultados mostrados a continuación se utilizó una búsqueda de manera secuencial, es decir, no se utilizó el índice LSH. Lo anterior debido a que la finalidad de esta sección es validar el funcionamiento únicamente del esquema de seguimiento y no el probar el desempeño del índice. Así, al no utilizar el índice LSH se eliminan los posibles problemas de desempeño ocasionados por el mismo. Se descartan los parámetros que determinan la cantidad de bits por vector a utilizar para calcular la función hash, el número de variaciones por vector que se crean y la cantidad de mapas hash a utilizar ($howManyBits$, $bitVariations$ y $howManyMaps$ véase la Sección **??**). Por lo cual, los únicos parámetros relevantes para ésta sección serán k, $mode$, $windowMultiplierDefault$ (wmd, por conveniencia) y $distance$ puesto que nB y $windowSize$ tienen sus valores fijos en 17 y 43 respectivamente como se expuso en la Sección **??**.

A continuación se presentan algunos de los resultados de validación obtenidos. Los resultados mostrados fueron seleccionados para ejemplificar de manera evidente las diferencias en el comportamiento del algoritmo de seguimiento que resultan de la variación de los parámetros mencionados en el párrafo anterior. En cada caso se tomo una de las grabaciones con las que se cuenta y se realizaron modificaciones aleatorias a su huella de audio. Únicamente se hicieron 2 tipos de modificaciones a las huellas:

1. Se replicaron algunas filas para simular situaciones en las que el intérprete sostiene alguna nota o acorde por un tiempo determinado.

2. Se removieron algunas filas para simular situaciones en las que el intérprete se "salta" parte de la partitura.

En cada figura se indica la grabación utilizada, las modificaciones realizadas a

la huella, el resultado esperado y el resultado obtenido por el esquema de seguimiento implementado.

4.1.1. Variando el modo de seguimiento, *mode*

El parámetro *mode* determina la manera en la que se elige la siguiente posición de entre la lista de candidatos posibles. Los 4 modos implementados fueron expuestos a profundidad en la Sección **??**. En las Figuras **??**, **??**, **??** y **??** se presenta el comportamiento de cada uno de los diferentes modos implementados.

Se utilizó la grabación del *Etude* #16 la cual tiene una duración de 1 minuto con 11 segundos y su firma tiene 6126 líneas. Se replicó la fila 900 150 veces, la fila 1720 130 veces, las filas 2750 a 3850, 3400 a 3550 y 4650 a 4780 fueron removidas y, por último la fila 5300 se replicó 100 veces.

El resto de los parámetros se utilizaron con los siguientes valores: $k = 20$, $wmd = 4$, $distance = Levenshtein$.

El Modo 1 ejemplificado en la Figura **??** elige el candidato más cercano en tiempo por lo que favorece pasos pequeños, lo anterior se puede observar en dicha figura donde el ejemplo más claro se presenta entre los índices 3200-3700. El algoritmo elige posiciones demasiado cercanas por lo cual se aleja del resultado esperado.

El Modo 2 ejemplificado en la Figura **??** elige al candidato más cercano en distancia, ésto puede tener resultados no deseados como se puede corroborar en la misma figura en los índices 3400-3700. El algoritmo elige un candidato muy parecido pero demasiado lejano y se aleja del resultado esperado.

El Modo 3 ejemplificado en la Figura **??** elige al candidato considerado como más adecuado entre el más cercano en tiempo y el más cercano en distancia. Se observa en la misma figura que este modo tiene un comportamiento superior a los Modos 1 y 2.

El Modo 4 ejemplificado en la Figura **??** calcula una extrapolación lineal a partir de las últimas 2 posiciones elegidas y elige al candidato más cercano a la extrapolación calculada. Se observa en dicha figura que este modo tiende a favorecer los cambios poco bruscos y tiene un comportamiento adecuado pero no superior al Modo 3.

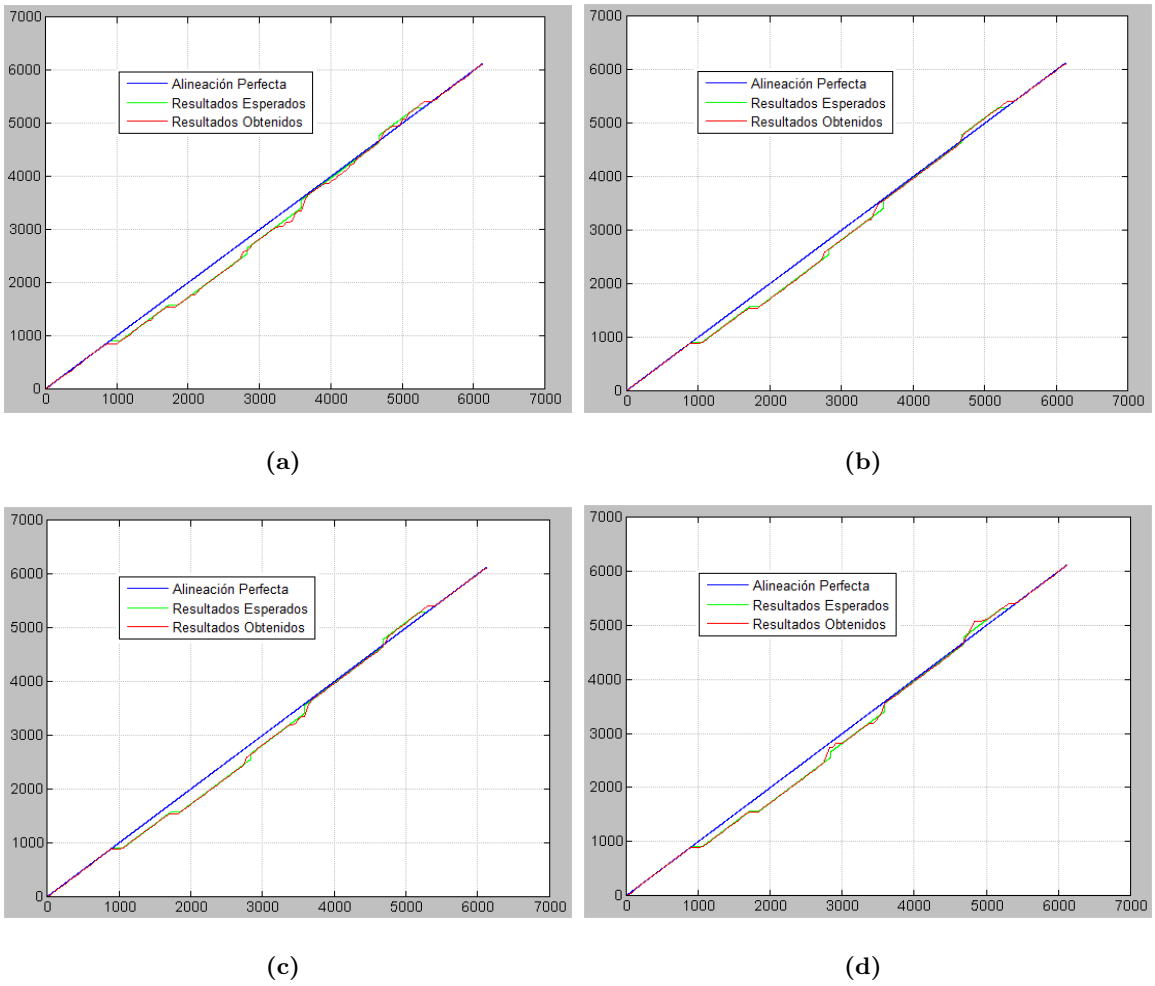


Figura 4.1: Comportamiento del seguimiento con los diferentes modos. (a).- Modo 1. (b).- Modo 2. (c).- Modo 3. (d).- Modo 4.

4.1.2. Variando el cálculo de distancia, *distance*

El parámetro *distance* determina cuál de las 3 distancias implementadas se utiliza para calcular la distancia entre las subfirmas. Las distancias que se utilizaron fueron Levenshtein, Hamming y LCS (véase la Sección **??**).

Para los experimentos mostrados a continuación se utiliza la misma grabación que en la sección anterior con las mismas modificaciones y con todos los mismos valores en todos los parámetros, a excepción del parámetro *mode* el cual se utiliza con un valor de 3 puesto que se observó en la sección previa que dicho modo es el que presenta un mejor comportamiento.

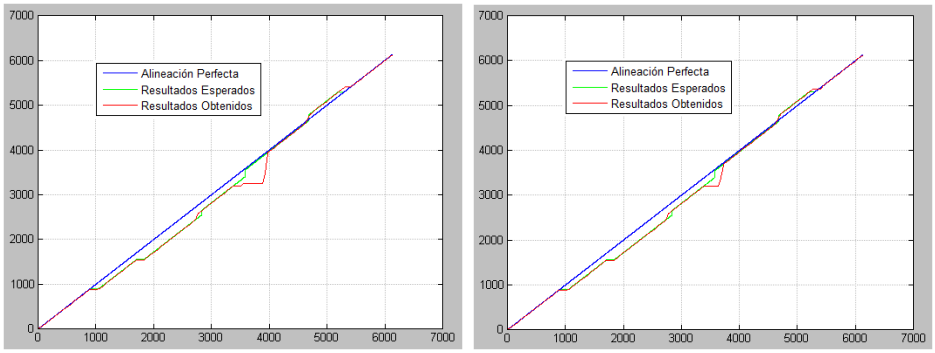
En las Figuras **??**, **??**, **??** se muestra como al utilizar distancias diferentes cálculos de distancia el comportamiento del algoritmo cambia puesto que las listas de candidatos obtenidas son diferentes.

4.1.3. Variando el tamaño de la lista de candidatos, k

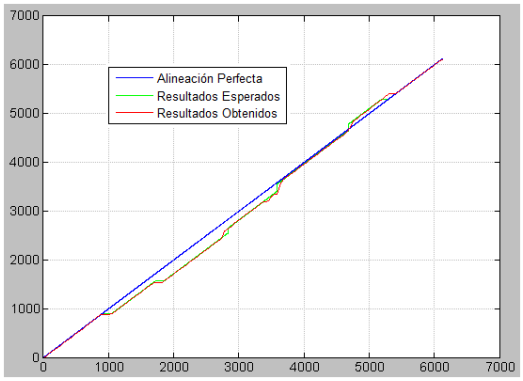
El parámetro k determina la cantidad de candidatos que se obtendrán para de entre ellos elegir la siguiente posición. Es evidente que si se tienen muy pocos candidatos para elegir será altamente probable que no se encuentre uno entre ellos que cumpla los requisitos del seguimiento (véase la Sección **??**). Por otro lado si se tienen demasiados candidatos la situación se revierte, es muy probable que se encuentre un candidato que cumpla los requisitos, sin embargo, dicho candidato podría resultar ser inadecuado debido a que podría ser muy diferente a la matriz que se busca.

Para ejemplificar ambos casos se utilizan 2 ejemplos distintos. Para el primer caso (k bajo) se utilizó el *Etude* #10 que tiene una duración de 1 minuto y 27 segundos, por lo cual su firma tiene 7554 líneas. La fila 1600 se replicó 120 veces, las filas 2800-2920 y 3500-3600 se eliminaron, las filas 4100 y 5600 se replicaron 100 y 150 veces respectivamente y, por último, las filas 6300-6450 se removieron. Se utilizaron los siguientes valores para el resto de los parámetros: $wmd = 4$, $distance = Levenshtein$, $mode = 3$.

En la Figura **??** se presenta el comportamiento del seguimiento con un valor de k adecuado, en este caso, $k = 20$. Así mismo, en la Figura **??** se presenta el comportamiento



(a) **(b)**



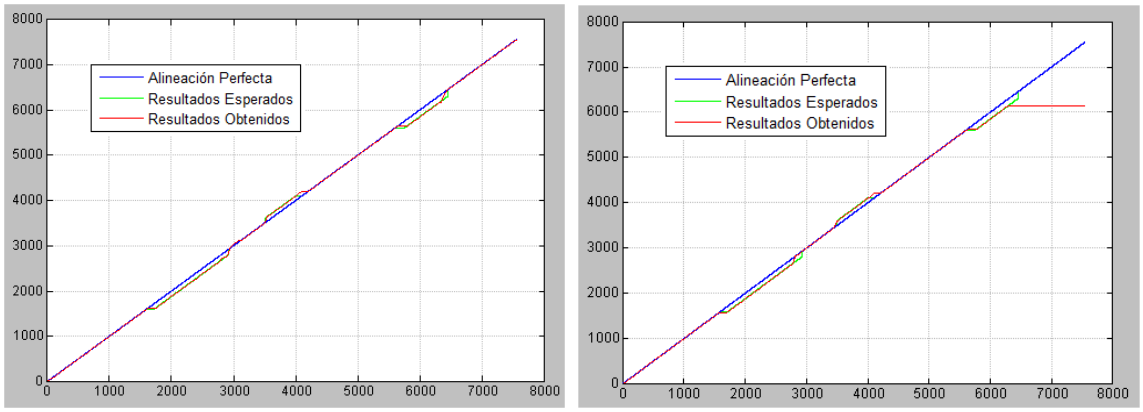
(c)

Figura 4.2: Comportamiento del seguimiento con las diferentes distancias (a).- Hamming. (b).- LCS. (c).- Levenshtein.

del seguimiento con un valor de k demasiado bajo, en este caso, $k = 10$. Se puede observar en esta figura que al final de la interpretación el algoritmo es incapaz de encontrar un candidato adecuado para la siguiente posición por lo cual se estanca y permanece en la misma posición hasta el final.

Para el segundo caso (k alto) se utilizó la *Ballade* #14 con una duración de 2 minutos con 31 segundos y firma de 13040 líneas. La línea 2400 se replicó 100 veces, las filas 3700-3820, 5200-5350 y 7000-7100 se eliminaron, por último, las filas 9700 y 11400 se replicaron 120 y 150 veces respectivamente. Se utilizaron los siguientes valores para el resto de los parámetros: $wmd = 4$, $distance = Levenshtein$, $mode = 3$.

En la Figura **??** se presenta el comportamiento del seguimiento con un valor de k



(a) **(b)**

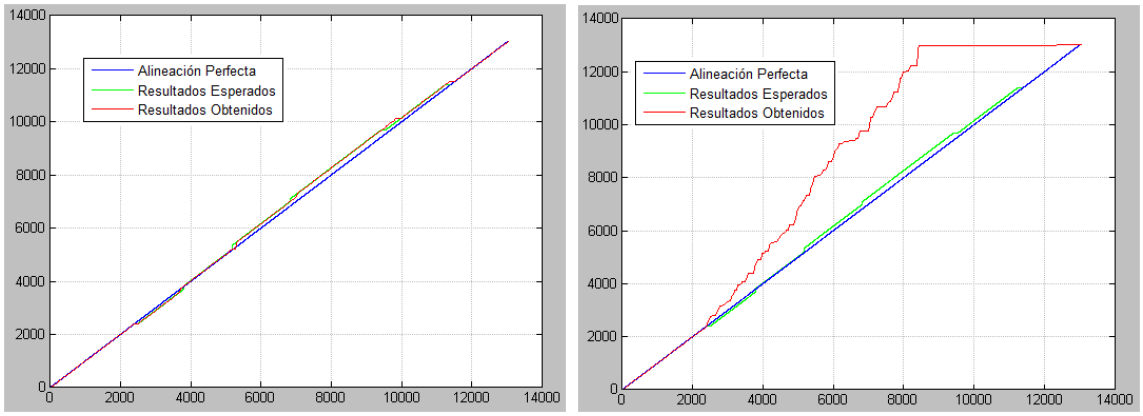
Figura 4.3: Primer ejemplo del comportamiento del seguimiento con diferentes valores de k (a).- Valor Adecuado. (b).- Valor demasiado bajo.

adecuado, en este caso, $k = 35$. De la misma forma, en la Figura **??** se presenta el comportamiento del seguimiento con un valor de k demasiado alto, en este caso, $k = 100$. Se puede observar en la figura que al tener tantos candidatos el algoritmo siempre encuentra alguno que cumple con los requisitos y avanza rápidamente llegando al final de la interpretación antes de lo adecuado. Como se mencionó en las Secciones **??** y **??**, este efecto puede mitigarse con el parámetro $windowMultiplierDefault$ pero podría traer otros efectos negativos en el comportamiento.

En los experimentos realizados con las grabaciones mencionadas se encontró que generalmente se puede elegir un valor de k adecuado en el rango de $\approx [15-45]$ dependiendo del tamaño de la interpretación. Para interpretaciones más largas se requiere un valor k más grande. Aunque valores menores o mayores a este rango podrían tener un funcionamiento adecuado dependiendo del caso en particular.

4.1.4. Variando el tamaño de los saltos, $windowMultipierDefault$

El parámetro $windowMultiplierDefault$ o wmd establece un límite superior para la elección de la siguiente posición. En la Sección **??** se brinda una explicación a detalle del funcionamiento de este parámetro. En resumen, determina que tan lejos (en tiempo) puede



(a) **(b)**

Figura 4.4: Segundo ejemplo del comportamiento del seguimiento con diferentes valores de k (a).- Valor Adecuado. (b).- Valor demasiado alto.

estar un candidato para que pueda ser considerado como siguiente posición.

Para los experimentos siguientes se utilizó el *Etude* #5 con duración de 1 minuto con 10 segundos y cuya firma tiene 6027 líneas. La fila 1000 se replicó 100 veces y la fila 2450 se replicó 150 veces. A su vez, se eliminaron las filas 3800-3900 y 5100-5250. Se utilizaron los siguientes valores para el resto de los parámetros: $k = 20$, $distance = Levenshtein$, $mode = 3$.

En la Figura **??** se utiliza un valor *wmd* adecuado. En particular se utilizó $wmd = 4$. Como se puede observar en la figura, el valor utilizado permite un alineamiento adecuado de la interpretación. Ambos "saltos" creados artificialmente se solventan de manera adecuada.

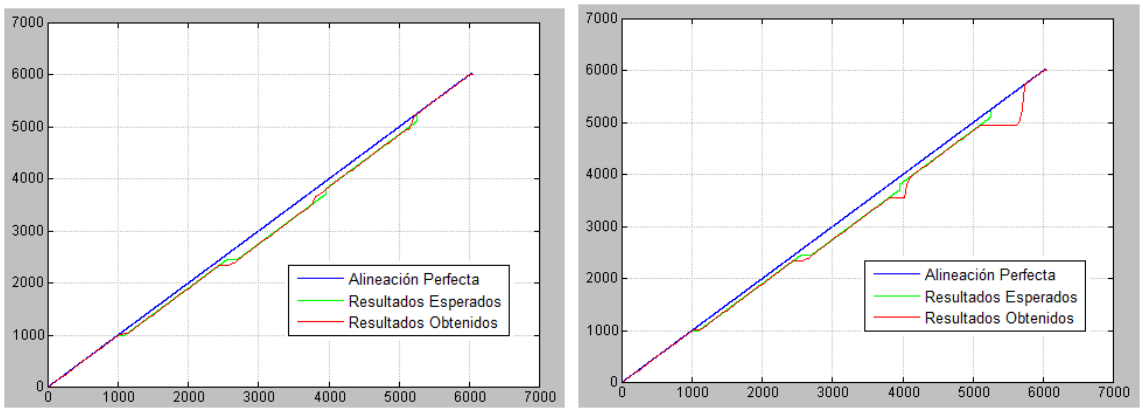
En la Figura **??** se observa el resultado de utilizar un valor *wmd* demasiado bajo. En este caso en particular se utilizó un valor de $wmd = 2$. El seguimiento resulta adecuado hasta el punto donde aparece el primer "salto". Puesto que el "salto" generado es de 100 posiciónes (se borraron las filas 3800-3900) y $wmd = 2$, el algoritmo no encuentra un candidato que cumpla con los requisitos:

$$windowMultiplier * windowSize = 2 * 43 = 86 < 100$$

Debido a ésto, se mantiene en la misma posición hasta que el valor de $windowMultiplier$

a aumentado lo suficiente para encontrar un candidato adecuado. Ésto ocurre hasta el índice 4100 aproximadamente.

Lo mismo sucede, de manera aún más evidente, con el segundo salto de 150 posiciones (se borró 5100-5250) el cuál se recupera hasta el índice 5700 aproximadamente.



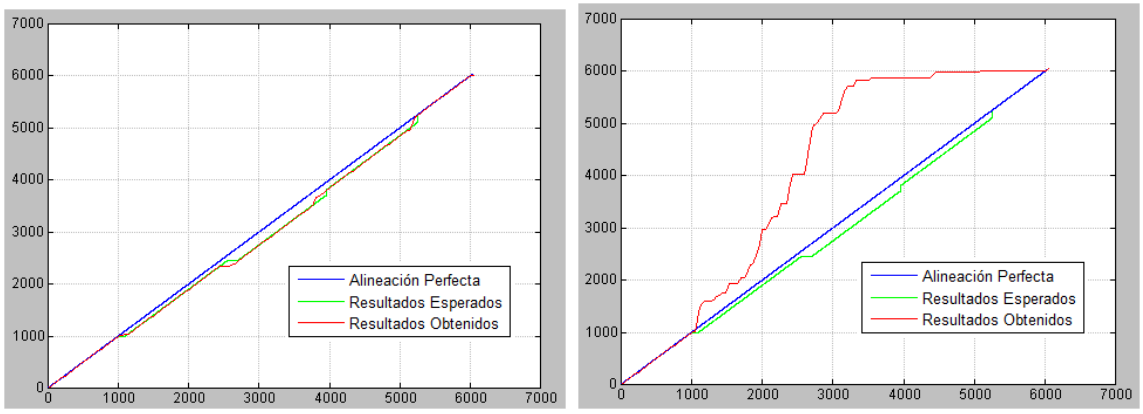
(a) **(b)**

Figura 4.5: Segundo ejemplo del comportamiento del seguimiento con diferentes valores de wmd (a).- Valor Adecuado. (b).- Valor demasiado bajo.

En la Figura **??** se observa el resultado de utilizar un valor de *wmd* demasiado alto. En este caso en particular se utilizó $wmd = 9$ lo cual permite que la siguiente posición pueda ser elegida 387 posiciones adelante de la posición anterior (≈ 4.5 segundos). Adicionalmente se aumentó el valor de k de 20 a 40 para proveer al algoritmo con más posibles candidatos y hacer el efecto más evidente.

Se observa en la Figura **??** que en cuanto se llega a la primera modificación (fila 1000 replicada 100 veces) el algoritmo en lugar de permanecer en el mismo índice, como se esperaría, encuentra un candidato muy alejado en tiempo de la posición anterior que sin embargo cumple los requisitos para ser elegido como posición siguiente debido al valor tan alto de *wmd*. La situación anterior se repite a lo largo del seguimiento lo cual evita que éste se recupere y el resultado es evidentemente pésimo.

El parámetro *wmd* es entonces sumamente importante y una elección adecuada del mismo puede ser la diferencia entre un seguimiento apropiado y uno muy malo. Elegir



(a) **(b)**

Figura 4.6: Segundo ejemplo del comportamiento del seguimiento con diferentes valores de wmd (a).- Valor Adecuado. (b).- Valor demasiado alto.

un valor demasiado bajo puede ocasionar que algunos de los saltos en la interpretación no sean manejados de manera adecuada; y elegir un valor demasiado alto (en particular en conjunto con un valor de k alto) ocasiona que el algoritmo avance en lugar de mantenerse donde debería.

En los experimentos realizados a lo largo de esta sección se conoce de antemano las modificaciones realizadas a las huellas de audio y, por lo tanto, se puede elegir siempre un valor para *wmd* que sea adecuado. Este no será el caso cuando se comparen 2 interpretaciones diferentes así que se deberá elegir un valor *wmd* basándose en conocimientos empíricos u otra información que sea relevante como puede ser la duración de ambas interpretaciones o las habilidades de los intérpretes.

Para el presente trabajo, y debido a que las grabaciones utilizadas fueron realizadas por pianistas profesionales, se puede estimar que las interpretaciones son lo suficientemente competentes y no existirán saltos mayores a 3 segundos ($wmd <= 6$). Por lo tanto, se utilizará usualmente un valor de *wmd* en el rango $\{3, .., , 6\}$.

Para un recordatorio del funcionamiento e interacción entre los parámetros *wmd*, $windowMultiplier$ y $windowSize$ véase la Sección **??**.

4.1.5. Resumen de la validación

En base a los resultados obtenidos en los experimentos se concluye que el algoritmo de seguimiento propuesto tiene un comportamiento adecuado, supeditado a la elección adecuada de sus parámetros.

En la Tabla **??** se presenta un resumen de los valores recomendados para los parámetros del algoritmo.

Parámetro	Valores Recomendados
k	$\{n \mid n \in N, 15 \leq n \leq 45\}$
distance	Levenshtein, LCS
mode	3, 4
wmd	$\{n \mid n \in N, 3 \leq n \leq 6\}$

Tabla 4.1: Valores recomendados para los parámetros de seguimiento

4.2. Resultados: Búsqueda Secuencial

En esta sección se presentan los resultados obtenidos al realizar el alineamiento de dos interpretaciones distintas, utilizando el algoritmo de seguimiento propuesto con búsqueda secuencial, cuya validación se presentó en la sección anterior.

Estos resultados se tomarán como referencia en la Sección **??** en la cual se evalúa el desempeño del índice LSH.

De todos los experimentos realizados, se eligieron los ejemplos descritos en las subsecciones **??** a **??** debido a que en ellos se utilizan grabaciones de diferentes duraciones y características. Por tanto, ayudan a visualizar el comportamiento del algoritmo en diversas posibles situaciones que se pueden presentar.

Para cada ejemplo se mencionan las grabaciones que fueron utilizadas y el número de comparaciones entre matrices que se realizan por cada subfirma a buscar. Se incluyen dos gráficas por cada ejemplo puesto que para cada par de grabaciones utilizado se realizaron 2 experimentos: primero utilizando una de las interpretaciones como interpretación de referencia y posteriormente utilizando la otra.

Los valores de k y wmd utilizados se tomaron de los rangos recomendados en la

Sección **??**. La distancia de Levenshtein y el Modo 3 para elección de la posición siguiente fueron utilizados puesto que mostraron los mejores resultados durante la validación y los experimentos realizados.

Todas las formas de onda de las grabaciones usadas, así como sus duraciones exactas, se pueden consultar en el Apéndice **??** con el fin de que el lector pueda constatar por sí mismo las características de cada interpretación que se mencionan en los ejemplos presentados.

4.2.1. Ejemplo 1

Para este ejemplo se utilizaron grabaciones que tienen aproximadamente la misma duración y cuyos intérpretes tienen ritmos similares por lo cual su alineamiento debería ser casi perfecto, lo que gráficamente se observaría como una línea a 45 grados. Las grabaciones utilizadas fueron los Etudes #4 (**??**) y (**??**), ambas con una duración de 1 minuto y 27 segundos.

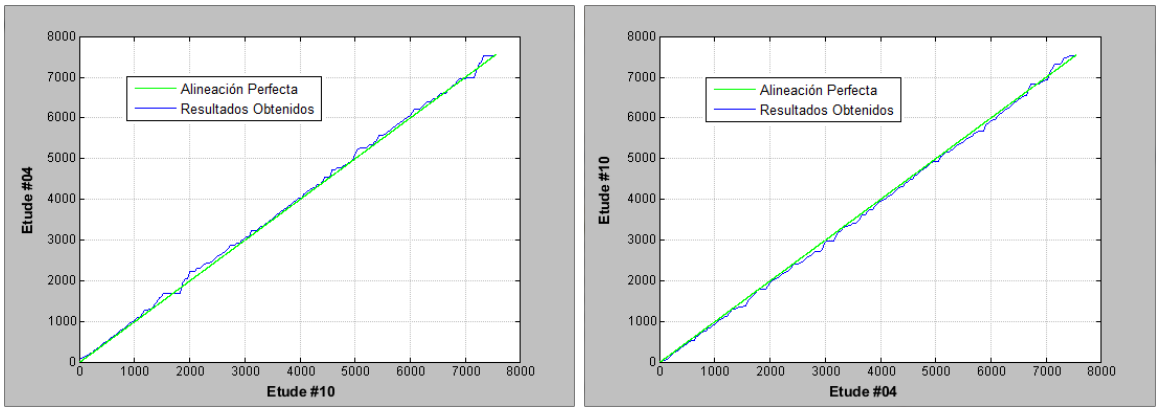
La Figura **??**(a) presenta el resultado del alineamiento utilizando el Etude #4 como referencia. Se observa que el alineamiento se mantiene muy ligeramente por encima de la línea a 45 grados que representa el alineamiento perfecto. Debido al tamaño de la firma del Etude #4, se realizan 7511 comparaciones por cada subfirma que se busca.

En la Figura **??**(b) se utiliza el Etude #10 como referencia y, como podría esperarse, el alineamiento se mantiene ligeramente por debajo de la línea a 45 grados. Se realizan 7505 comparaciones por subfirma. Estos resultados muestran que las interpretaciones son casi iguales y que su alineamiento es casi perfecto.

4.2.2. Ejemplo 2

Para este ejemplo se utilizaron los Etudes #20 (**??**) y #21 (**??**) cuyas grabaciones tienen una duración similar. El Etude #20 tiene una duración de 1 minuto y 30 segundos; mientras que el Etude #21 tiene una duración de 1 minuto y 29 segundos.

Lo anterior podría sugerir que simplemente existirá un defasamiento de 1 segundo entre ambas interpretaciones. Sin embargo, aunque ambas interpretaciones comienzan de manera similar, el intérprete del Etude #21 tiene un ritmo ligéramente más rápido que el



(a) **(b)**

Figura 4.7: Alineamiento de los Etudes #4 y #10 (a).- Etude #04 utilizado como referencia. (b).- Etude #10 utilizado como referencia.

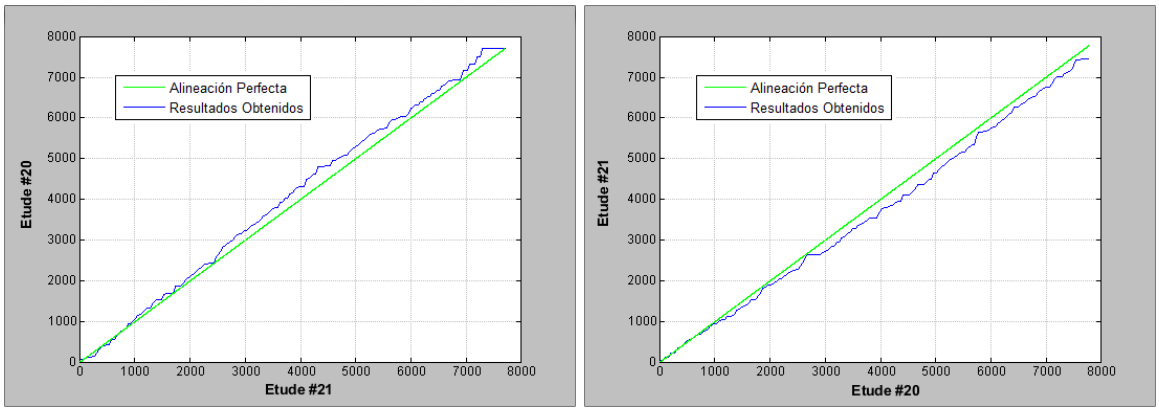
del Etude #20. De manera que para el Etude #21 la interpretación termina alrededor del 1:26 mientras que en el Etude #20 termina en el segundo 1:28. Ésto se puede apreciar en la Figura **??**.

En la Figura **??**(a), donde se utiliza el Etude #20 como referencia, se observa como el alineamiento se separa paulatinamente por encima de la línea a 45 grados, lo cual indica que la interpretación *online* (Etude #21) tiene un ritmo más rápido que la interpretación de referencia. Igualmente se puede observar en la parte superior derecha que se alcanza el final de la interpretación antes de llegar al final de la interpretación de referencia. En este caso se realizan 7732 comparaciones por subfirma.

En la Figura **??**(b), se utiliza el Etude #21 como referencia. Ésto produce un efecto inverso al caso anterior en el cual el alineamiento se separa paulatinamente pero por debajo de la línea a 45. Lo anterior indicando que la interpretación *online* tiene un ritmo más lento que la interpretación de referencia. Se realizan 7655 comparaciones por subfirma.

4.2.3. Ejemplo 3

Para este ejemplo se utilizaron las grabaciones #1 (**??**) y #2 (**??**) del Etude las cuales tienen duraciones muy diferentes. El Etude #1 tiene una duración de 1 minuto y 26



(a) **(b)**

Figura 4.8: Alineamiento de los Etudes #20 y #21 (a).- Etude #20 utilizado como referencia. (b).- Etude #21 utilizado como referencia.

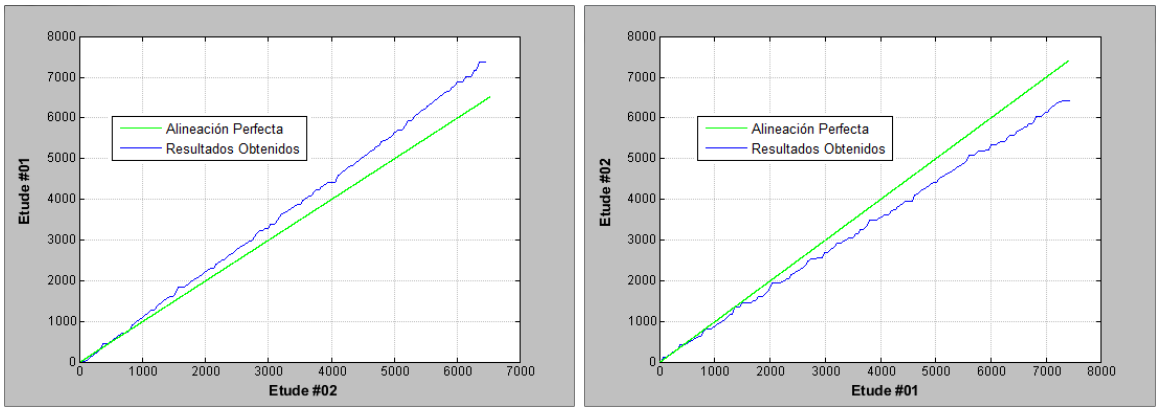
segundos, mientras que el Etude #2 tiene una duración de 1 minuto y 16 segundos.

Esta diferencia de 10 segundos entre las grabaciones nos indica que el intérprete del Etude #2 tiene un ritmo mucho más rápido que su contraparte del Etude #1. Las interpretaciones terminan aproximadamente en las marcas 1:24 y 1:15 respectivamente para Etude #1 y #2. Además, existe un silencio ligeramente más largo al inicio del Etude #2.

En la Figura **??**(a), donde se utiliza el Etude #1 como referencia, se aprecia como el alineamiento se separa rápidamente por encima de la línea a 45 grados y continua así hasta el final de la interpretación; corroborando así que el ritmo del intérprete del Etude #2 es mucho más rápido. Debido al tamaño de la firma del Etude #1 se realizan 7363 comparaciones por cada subfirma buscada.

En la Figura **??**(b) se observa el resultado de realizar el alineamiento de manera inversa (utilizando el Etude #2 como referencia). El resultado es el esperado ya que el alineamiento tiene un comportamiento opuesto al caso anterior. Para este caso, se realizan 6535 comparaciones por subfirma.

En ambos casos se puede observar que se alcanza un defasamiento de entre 9 y 10 segundos como era de suponerse debido a las características de ambas interpretacioens.



(a) **(b)**

Figura 4.9: Alineamiento de los Etudes #01 y #02 (a).- Etude #01 utilizado como referencia. (b).- Etude #02 utilizado como referencia.

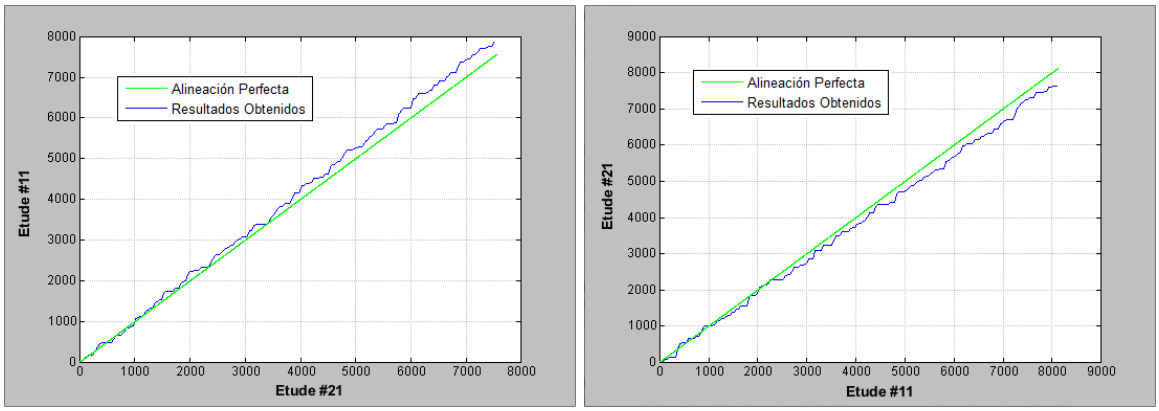
4.2.4. Ejemplo 4

Este es otro ejemplo en el cuál se utilizan dos interpretaciones con duraciones distintas. Se utilizaron los Etudes #11 (**??**) y #21 (**??**). El primero de ellos tiene una duración de 1 minuto y 34 segundos; el segundo de 1 minuto y 29 segundos.

Ambas interpretaciones tienen un silencio al inicio de entre 0 - 0.5 segundos y al final de entre 3 - 3.5 segundos. Por lo tanto se debería esperar un defasamiento de entre 4 y 5 segundos entre ellas.

En la Figura **??**(a), la cual utiliza el Etude #11 como referencia, se aprecia que el algoritmo tiene el comportamiento esperado: al ser el Etude #21 más corto que el #11 se supondría que el intérprete tiene un ritmo más rápido. Y así el seguimiento se separa gradualmente del alineamiento a 45 grados hasta llegar una diferencia de 4.5 segundos. En este caso se realizan 8078 comparaciones por cada subfirma.

De forma análoga, al utilizar el Etude #21 como referencia en la Figura **??**(b), el comportamiento es similar pero inverso puesto que ahora se utiliza la interpretación más corta como referencia. Se efectúan 7655 comparaciones por cada subfirma.



(a) **(b)**

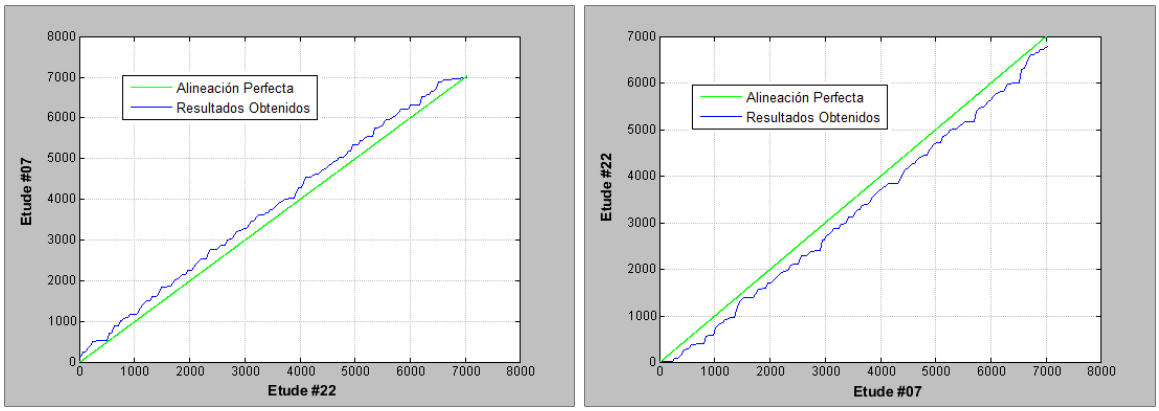
Figura 4.10: Alineamiento de los Etudes #11 y #21 (a).- Etude #11 utilizado como referencia. (b).- Etude #21 utilizado como referencia.

4.2.5. Ejemplo 5

El siguiente ejemplo es un caso bastante particular. Se utilizaron las grabaciones del Etude #7 (**??**) y #22 (**??**). Aparentemente, ambas interpretaciones deberían ser muy similares puesto que sus grabaciones tienen duraciones casi exactamente iguales (1 minuto y 26 segundos) y su alineamiento debería ser casi perfecto como el mostrado en **??**.

Sin embargo, el Etude #7 tiene la particularidad de tener un silencio bastante largo al inicio (entre 2 y 2.5 segundos aproximadamente); y de terminar la interpretación casi simultáneamente a la grabación, es decir, no tiene casi silencio al final. A su vez, el Etude #22 comienza la interpretación de manera casi inmediata pero contiene un silencio de entre 2 y 2.5 segundos al final de la grabación. Estas características se compensan entre sí ocasionando que el alineamiento entre ambas interpretaciones sea casi a 45 grados pero defasado de la linea de la alineación perfecta.

En la Figura **??**(a) se presenta el comportamiento del alineamiento usando el Etude #7 como referencia. Se aprecia que, como el Etude #22 no tiene el silencio inicial, existe un salto al principio del alineamiento. Posteriormente se puede entonces ver que después del salto el alineamiento se ajusta aproximadamente a una línea a 45 grados como se ilustra en la Figura **??**(a).



(a) **(b)**

Figura 4.11: Alineamiento de los Etudes #7 y #22 (a).- Etude #7 utilizado como referencia. (b).- Etude #22 utilizado como referencia.

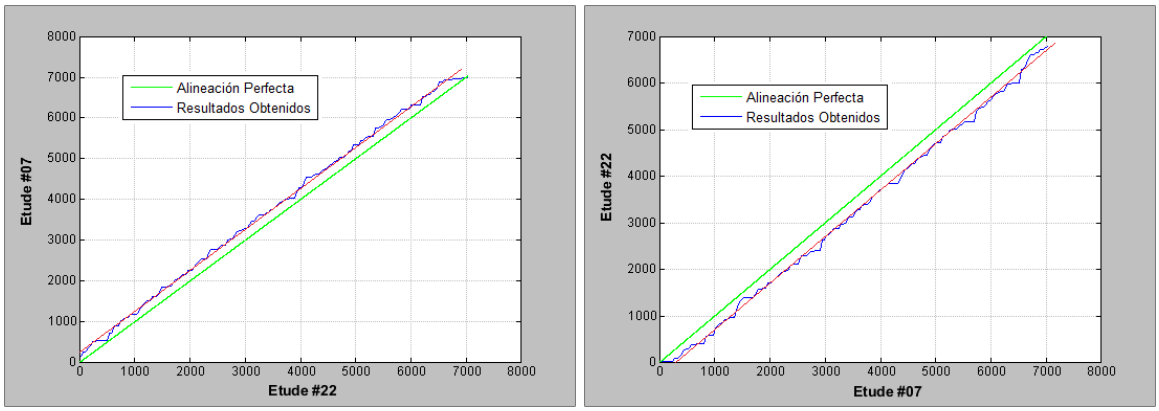
De igual forma, en la Figura **??**(b) se observa el comportamiento usando el Etude #22 como referencia. En este caso, el algoritmo se mantiene en la misma posición durante el silencio inical del Etude #7 y comienza a avanzar una vez que comienza la interpretación. Similarmente al caso anterior, el seguimiento se ajusta a una línea de 45 grados como se puede observar en la Figura **??**(b). Debido al tamaño de las firmas, se realizan 6998 y 6989 comparaciones en cada caso respectivamente por cada subfirma a buscar.

4.2.6. Ejemplo 6

Como último ejemplo se presenta un experimento que se realizó para probar el comportamiento del seguimiento implementado utilizando dos interpretaciones diferentes pero donde una de ellas ha sido modificada artificialmente como se realizó en la Sección **??**.

Se utilizaron los Etudes #4 (**??**) y #10 (**??**)puesto que tienen un alineamiento casi perfecto como se mostró en **??**. Se utilizó el Etude #10 como referencia y se modificó la firma del Etude #4 de la siguiente manera: se replicó 150 veces la fila 1200, se borraron las filas 2450-2600, se replicó 200 veces la fila 4500 y, por último, se borraron las filas 6300-6500.

En la Figura **??** se puede observar el resultado. El seguimiento reacciona de la manera esperada ante las modificaciones: se mantiene en la misma posición cuando se llega



(a) **(b)**

Figura 4.12: Alineamiento de los Etudes #7 y #22 con líneas ilustrativas (a).- Etude #7 utilizado como referencia. (b).- Etude #22 utilizado como referencia.

al índice 1200 del Etude #4; existe un salto al llegar aproximadamente al índice 2500 y nuevamente se mantiene en posición al llegar al índice 4500.

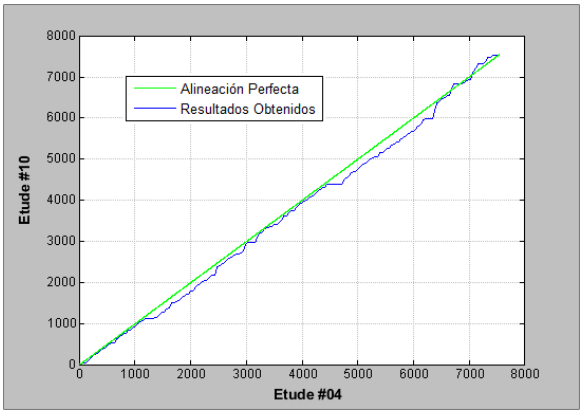


Figura 4.13: Alineamiento de los Etudes #4 y #10, utilizando #10 como referencia y modificando artificialmente la huella de #4.

El comportamiento presentado al llegar al índice 6300 se explica de la siguiente manera: se utilizaron los mismos parámetros que se utilizaron en el ejemplo de la Sección **??**. Entre ellos se definió un valor para $wmd = 4$ lo cual equivale a 2 segundos $(4 * 0.5s)$ o 172 $(4 * 43)$ posiciones. El salto del índice 6300 es de 200 posiciones, por lo tanto, excede el salto máximo. El índice elegido como siguiente posición se encuentra por tanto demasiado lejos

de la última posición elegida. El algoritmo entonces decide mantenerse en la misma posición y se incrementa el valor de *wm* (*windowMultiplier*, sección **??**). Lo anterior continúa hasta que se elige una posición siguiente adecuada (alrededor del índice 6400).

En resumen, se puede concluir que el seguimiento fue adecuado dadas las modificaciones que se efectuaron sobre la huella del Etude #4.

4.3. Resultados: Índice LSH

Esta sección se reserva para valorar el funcionamiento del índice LSH y justificar su utilización. Como se menciona en la Secciones **??** y **??** el funcionamiento del índice LSH depende principalmente de 2 cosas: la cantidad de instancias/mapas utilizados y la cantidad de bits que se utilizan para obtener la función hash.

Adicionalmente, debido a la manera en que se obtienen los índices candidato en la presente implementación (véase la Sección **??**), la cantidad de variaciones creadas por cada vector de la matriz binaria a buscar es otro factor a considerar.

En resumen son 3 los parámetros que determinarán el funcionamiento del índice LSH implementado:

1. *howManyMaps*
2. *howManyBits*
3. *bitVariations*

El efecto de aumentar o disminuir el valor de dichos parámetros se explica con mayor precisión en la Sección **??**. Un resumen breve se encuentra en la Tabla **??**.

El motivo de la utilización del índice LSH en lugar de utilizar la búsqueda secuencial consiste simplemente en obtener resultados equivalentes a ésta última, pero realizando un número considerablemente menor de comparaciones.

Para justificar el uso del índice, a continuación se muestran los resultados obtenidos en los diversos ejemplos que fueron explicados en la Sección **??** pero utilizando el índice LSH comprimido en lugar de la búsqueda secuencial.

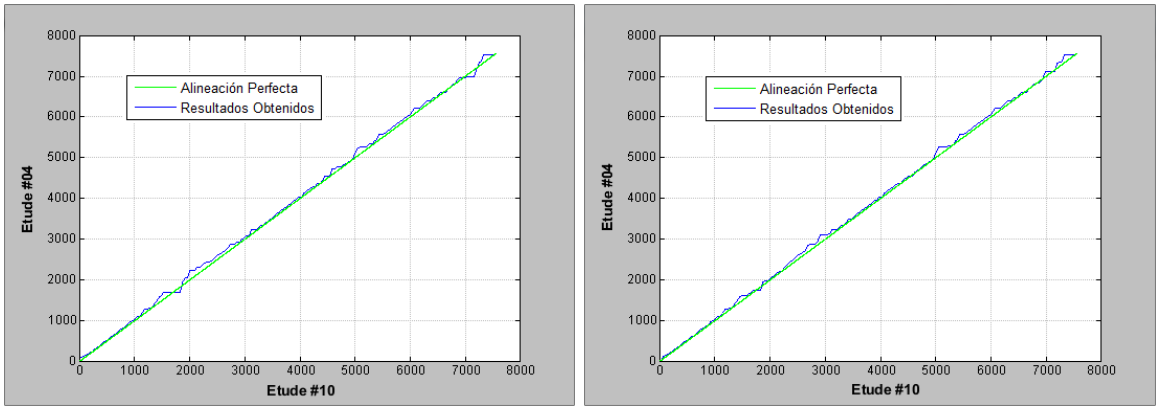
Parámetro	Aumentar	Disminuir
howManyMaps	Aumenta la cantidad de candidatos	Disminuye la cantidad de candidatos
howManyBits	Disminuye la cantidad de candidatos	Aumenta la cantidad de candidatos
bitVariations	Aumenta en gran medida la cantidad de candidatos	Disminuye en gran medida la cantidad de candidatos

Tabla 4.2: Resumen efectos de los parámetros LSH

En la Tabla **??** se presenta un condensado de los resultados obtenidos. En cada caso se presenta el número de ejemplo, la cantidad de comparaciones por subfirma que se efectúan utilizando búsqueda secuencial, el promedio de comparaciones realizadas con el índice LSH y los valores utilizados para los parámetros $howManyMaps$, $howManyBits$ y $bitVariations$. Así como una gráfica con los alineamientos obtenidos lado a lado usando búsqueda secuencial y el búsqueda con el índice LSH con el fin de que puedan ser comparados visualmente.

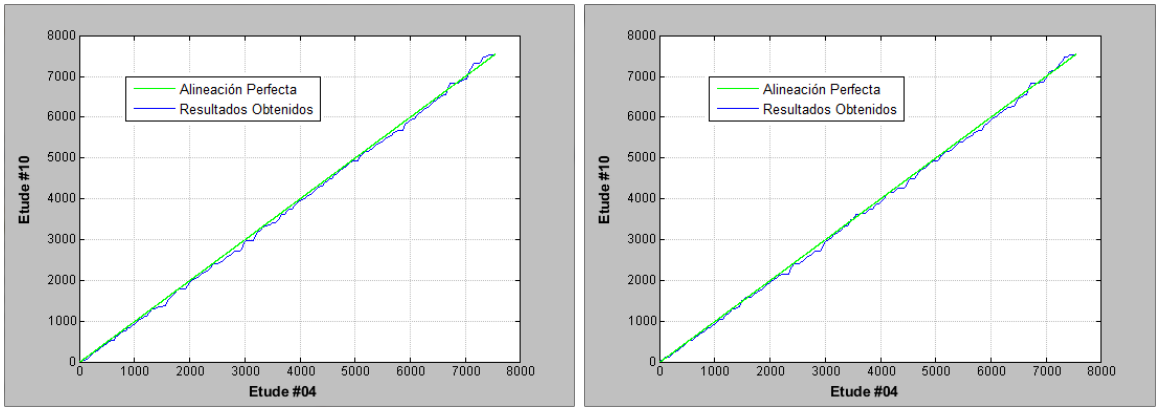
Número Ejemplo	**Figura**	**Secuencial**	**LSH**	**Reducción**	**hmm**	**hmb**	**vB**
Ej.1 Caso (a)	**??**	7511	1990.437	74 %	60	14	1
Ej.1 Caso (b)	**??**	7505	1925.38	75 %	60	14	1
Ej.2 Caso (a)	**??**	7732	2197.638	72 %	80	14	1
Ej.2 Caso (b)	**??**	7655	2535.077	67 %	25	13	1
Ej.3 Caso (a)	**??**	7363	2052.34	73 %	70	14	1
Ej.3 Caso (b)	**??**	6535	2358.225	64 %	30	13	1
Ej.4 Caso (a)	**??**	8078	2246.027	73 %	70	14	1
Ej.4 Caso (b)	**??**	7655	1969.423	75 %	60	14	1
Ej.5 Caso (a)	**??**	6998	2717.079	62 %	40	13	1
Ej.5 Caso (b)	**??**	6989	2492.256	65 %	30	13	1
Ej.6	**??**	7511	2257.937	70 %	80	14	1

Tabla 4.3: Resultados con el Índice LSH.



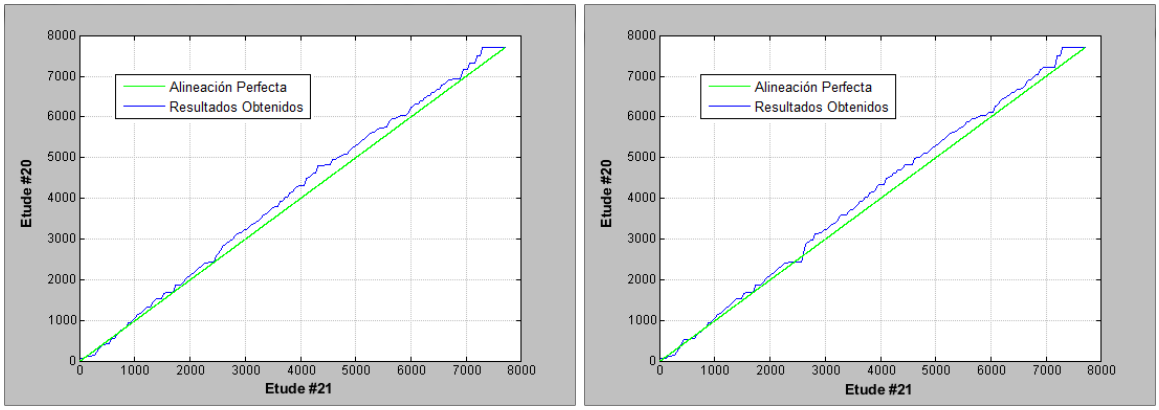
(a) **(b)**

Figura 4.14: Alineamiento de los Etudes #4 y #10 utilizando #4 como referencia (a).- Búsqueda Secuencial. (b).- Índice LSH.



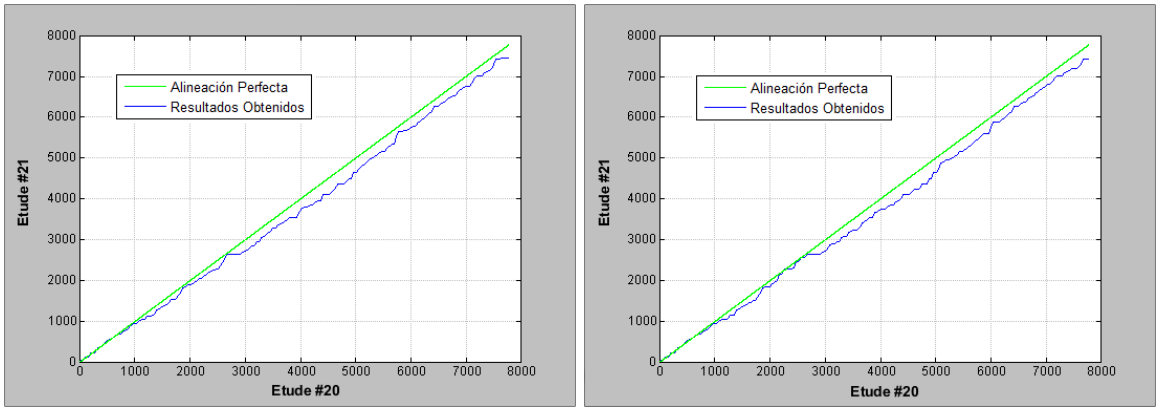
(a) **(b)**

Figura 4.15: Alineamiento de los Etudes #4 y #10 utilizando #10 como referencia (a).- Búsqueda Secuencial. (b).- Índice LSH.



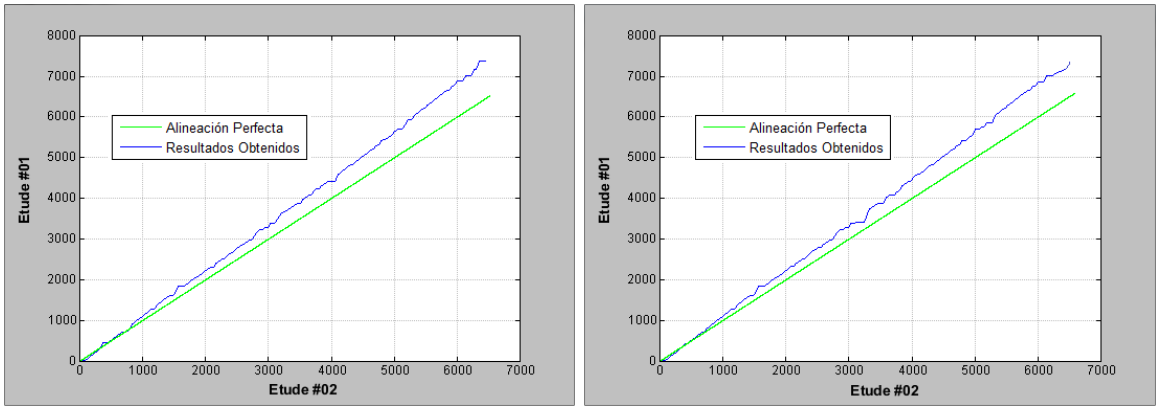
(a) **(b)**

Figura 4.16: Alineamiento de los Etudes #20 y #21 utilizando #20 como referencia (a).- Búsqueda Secuencial. (b).- Índice LSH.



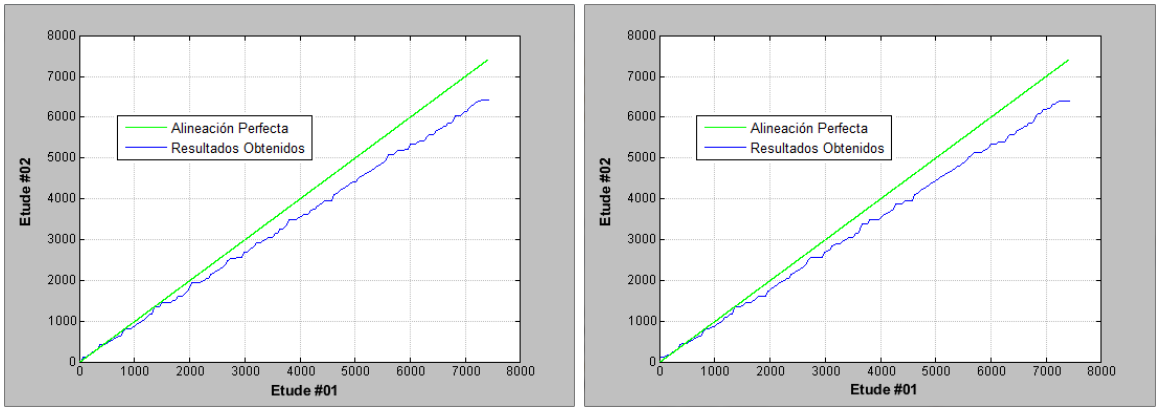
(a) **(b)**

Figura 4.17: Alineamiento de los Etudes #20 y #21 utilizando #21 como referencia (a).- Búsqueda Secuencial. (b).- Índice LSH.



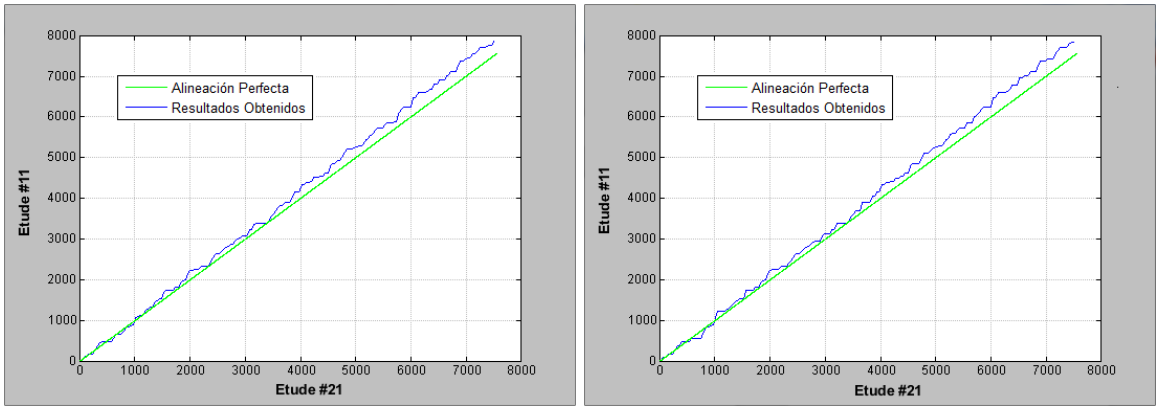
(a) **(b)**

Figura 4.18: Alineamiento de los Etudes #1 y #2 utilizando #1 como referencia (a).- Búsqueda Secuencial. (b).- Índice LSH.



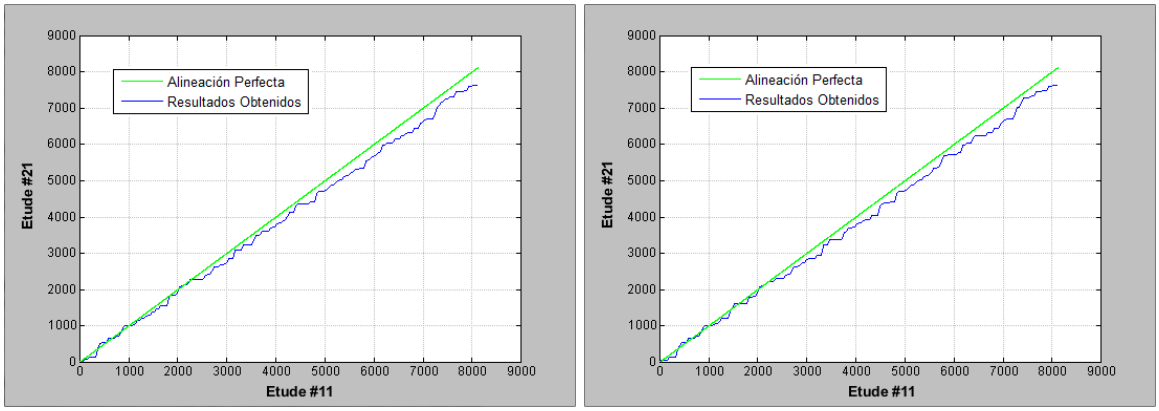
(a) **(b)**

Figura 4.19: Alineamiento de los Etudes #1 y #2 utilizando #2 como referencia (a).- Búsqueda Secuencial. (b).- Índice LSH.



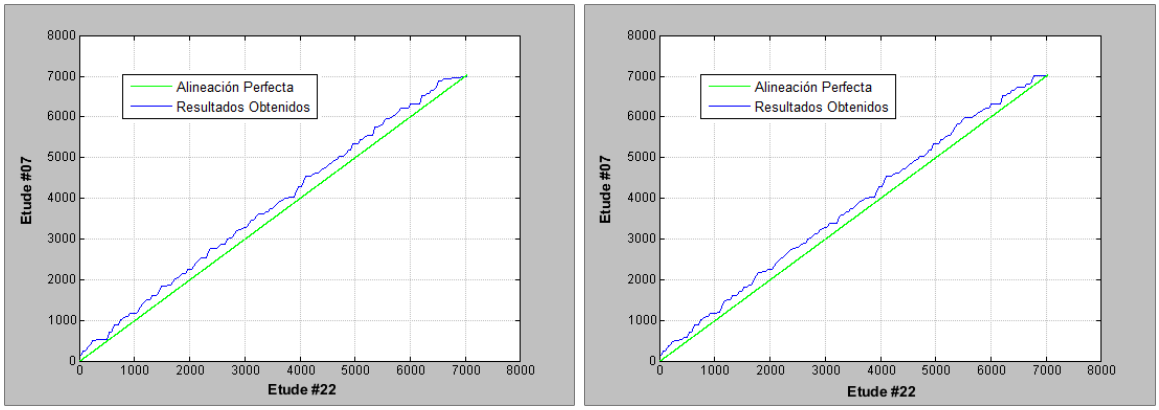
(a) **(b)**

Figura 4.20: Alineamiento de los Etudes #11 y #21 utilizando #11 como referencia (a).- Búsqueda Secuencial. (b).- Índice LSH.



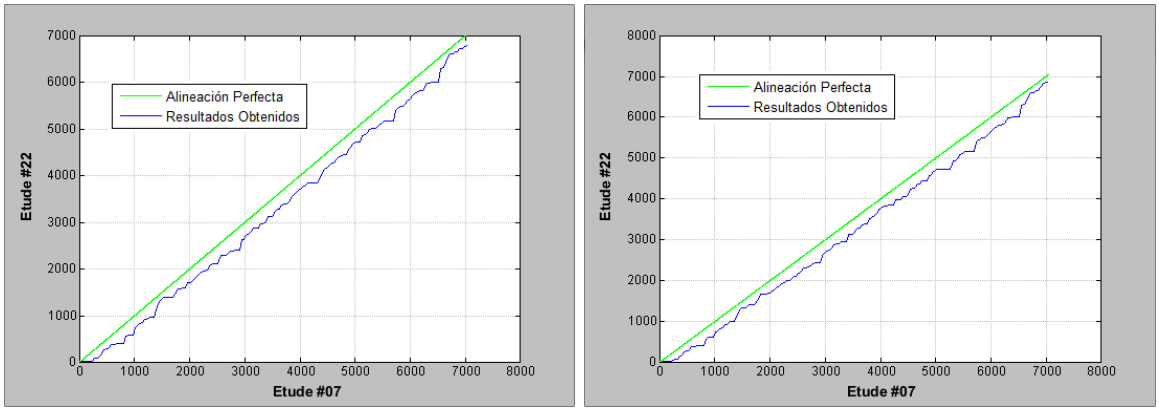
(a) **(b)**

Figura 4.21: Alineamiento de los Etudes #11 y #21 utilizando #21 como referencia (a).- Búsqueda Secuencial. (b).- Índice LSH.



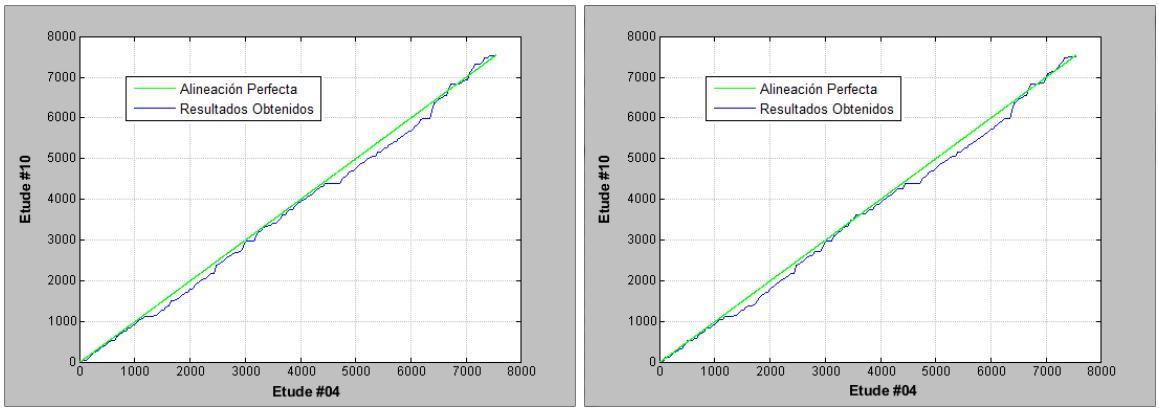
(a) **(b)**

Figura 4.22: Alineamiento de los Etudes #7 y #22 utilizando #7 como referencia (a).- Búsqueda Secuencial. (b).- Índice LSH.



(a) **(b)**

Figura 4.23: Alineamiento de los Etudes #7 y #22 utilizando #22 como referencia (a).- Búsqueda Secuencial. (b).- Índice LSH.



(a) **(b)**

Figura 4.24: Alineamiento de los Etudes #04 y #10 utilizando #10 como referencia y modificando artificialmente la huella de #4. (a).- Búsqueda Secuencial. (b).- Índice LSH.

4.4. Notas Finales

4.4.1. Notas Resultados LSH

- Todos los ejemplos presentados en la Sección **??** fueron calibrados para que los resultados fueran lo más parecido posible a los obtenidos con búsqueda secuencial pero tratando, a su vez, de reducir la cantidad de comparaciones realizadas con el fin de justificar el uso del índice. El número de comparaciones necesarias puede ser reducido aún más si se sacrifica un poco la similitud entre ellos.

- Como los bits a utilizar por cada instancia se generan aleatoriamente cada que se ejecuta el algoritmo. Dos ejecuciones con los mismos parámetros pueden reportar resultados ligeramente distintos.

- La importancia de utilizar el índice aumenta considerablemente conforme más largas son las interpretaciones. En general, se observó una reducción de entre 60 % y 75 % en la cantidad de comparaciones aunque este porcentaje puede ser mayor para interpretaciones con una duración mayor.

- Una reducción del 60 % podría parecer poco para justificar la creación del índice no obstante, como se mencionó anteriormente, se utiliza la distancia de Levenshtein, la cual es sumamente costosa de calcular. Por lo tanto, una reducción de incluso el 60 % en el número de comparaciones conlleva un efecto dramático en tiempo de ejecución, lo cual podría permitir que el algortimo pueda ser utilizado en tiempo real.

- La cantidad de mapas hash utilizada en los experimentos mostrados en la Sección **??** podría parecer excesiva, especialmente considerando los requerimientos de memoria. Sin embargo no es una cuestión relevante en nuestro problema por las siguientes razones:

  - En el índice LSH creado únicamente se indexarán los vectores de características de una de las interpretaciones a la vez y las interpretaciones utilizadas en los experimentos tienen de entre 6000 y 13000 vectores de características únicamente.

- El número de bits utilizado para crear el hash en los experimentos fue grande (13 o 14 bits de los 17 disponibles) lo cual parece implicar que los mapas serán bastante grandes con 2^{13} (8192) o 2^{14} (16384) posibles cubos. No obstante, incluso si se tomara una interpretación de 13000 sólamente si todos los hash creados fueran diferentes se crearían 13000 cubos, cada uno con un elemento. Ese jamás es el caso. Para la interpretación más larga utilizada, la Ballade #14 con una duración de 2 minutos y 31 segundos y cuya firma tiene 13040 líneas; utilizando 14 bits los mapas tienen 8300 cubos en promedio. Para un Etude como el #4 con una huella de 7548 líneas, el número de cubos es de 4500 únicamente.
- Utilizar un número alto de bits para crear el hash ocasiona además que los cubos tengan pocos elementos. Lo anterior tiene como consecuencia que se revisen pocos candidatos por mapa.
- De esta manera el número de bits utilizado en conjunto con la cantidad de mapas permite obtener los resultados reportados.
- La opción opuesta sería utilizar pocos bits para la creación de los hashes. De esa manera los cubos estarían más poblados y se utilizarían menos mapas para contrarrestar el número de comparaciones. Esta opción tiene un desempeño menos deseable debido a que la selección de las bandas que se utilizarán para crear el hash se hace al azar y ésto podría ocacionar que bandas con información importante no se tomen en cuenta.

4.4.2. Notas Adicionales

Se intentó acelerar el algoritmo limitando la revisión de posibles candidatos a aquellos que fueran mayores o iguales a la posición previa. Para la búsqueda secuencial se comenzaba la comparación entre matrices a partir de la posición indicada como anterior y para el índice LSH únicamente se revisaban candidatos superiores a la posición anterior. Lo anterior tuvo resultados contraproducentes puesto que las listas de vecinos más cercanos se llenaban con candidatos que no necesariamente tenían las matrices más parecidas. Ya que los verdaderos vecinos más cercanos habían sido descartados por encontrarse por detrás de

la posición anterior. Por ejemplo, si en determinado momento el algoritmo no encuentra un candidato dentro de la lista de vecinos más cercanos que cumpla con las restricciones, deberá optar por mantenerse en la misma posición. Si la lista de vecinos más cercanos está llena de candidatos que se encuentran por detrás de la posición anterior, ninguno de ellos cumple las restricciones y el algoritmo tomará dicha decisión. Si se evita revisar candidatos por detras de la posición anterior, el caso presentado jamás ocurrirá. En su lugar, la lista se llenará de otros candidatos que no deberían entrar a la lista. Ésto provocaría que el algoritmo avanzara cuando en realidad debería mantenerse.

4.5. Resumen de Capítulo

En este capítulo se presentó el esquema de validación diseñado para evaluar y validar el seguidor. La validación se basó en la utilización de firmas de audio modificadas en puntos específicos de manera que se puede comprobar si el seguimiento se comporta de la manera esperada. Posteriormente se presentaron los resultados obtenidos por el seguidor utilizando una búsqueda secuencial para la obtención de los posibles candidatos. Se observó que el seguidor tiene un comportamiento adecuado en los diferentes ejemplos presentados de acuerdo a las características de las interpretaciones utilizadas. Por último, se expusieron los resultados obtenidos al utilizar el índice LSH. Se corroboró que mediante la utilización del índice se reduce significativamente (60 %-75 %) la cantidad de comparaciones entre subfirmas realizadas, manteniendo un seguimiento equivalente al obtenido con la búsqueda secuencial. Habiendo presentado los resultados, el enfoque del siguiente capítulo será el discutir las conclusiones a las que se arrivó y los posibles trabajos futuros que pueden surgir del presente proyecto.

Capítulo 5

Conclusiones

5.1. Conclusiones Generales

El seguimiento de audio o alineamiento de interpretaciones es un problema que ha sido abordado de muchas maneras diferentes. En esta tesis se presentó un esquema que utiliza la huella de audio basada en la entropía presentada en (**?**) y el índice de búsqueda aproximada basado en *Locality Sensitive Hashing* (LSH) presentado originalmente en (**?**) y comprimido utilizando bitmaps en (**?**).

Se trabajó con grabaciones de 22 diferentes interpretaciones en piano de 2 piezas de Frédéric Chopin, las cuales han sido utilizadas en trabajos previos de índole similar al presente como (**?**).

Para realizar el alineamiento de 2 interpretaciones, se obtuvo la firma de cada una y se utilizó una de ellas como referencia para buscar la posición correspondiente a una subfirma equivalente a 0.5 segundos de audio de la otra interpretación.

Se validó el funcionamiento del algoritmo de seguimiento con experimentos realizados utilizando huellas de audio que fueron alteradas artificialmente en puntos específicos.

Al utilizar búsqueda secuencial se obtuvieron resultados satisfactorios en el alineamiento puesto que éstos coinciden con las características y particularidades de las interpretaciones comparadas.

Con la utilización del índice LSH comprimido se logró mantener un alineamiento

equivalente al obtenido con búsqueda secuencial reduciendo además, de manera considerable, la cantidad de comparaciones efectuadas por matriz.

Por las razones anteriores, se puede concluir que se cumplió con el objetivo planteado por el presente trabajo puesto que se implementó un algoritmo capaz de realizar el alineamiento entre 2 interpretaciones distintas de la misma pieza musical.

5.2. Conclusiones Específicas

1. Es evidente a partir de los resultados y explicaciones de los ejemplos presentados durante la sección **??**, que el algoritmo de seguimiento implementado con búsqueda secuencial muestra un comportamiento adecuado ante las características de las grabaciones utilizadas.

2. Con la implementación del índice LSH comprimido se logró una gran disminución en la cantidad de cálculos de comparación de subfirmas respecto a la búsquda secuencial. En los experimentos presentados se tienen reducciones de entre el 60 % y el 75 % y éste porcentaje puede ser mayor en interpretaciones con mayor duración.

3. La implementación de seguimiento expuesta puede ser utilizada en aplicaciones que requieran del alineamiento de interpretaciones musicales como podría ser un sistema de partitura automática para los intérpretes. No obstante, es todavía necesario realizar más experimentos para determinar la exactitud y viabilidad de la presente implementación en escenarios reales.

5.3. Trabajos Futuros

Como se mencionó en la sección anterior, aún existen diversas áreas de experimentación y expansión del trabajo que se presenta. A continuación se enumeran algunos de los posibles trabajos a futuro:

1. Al ser el presente trabajo uno de los primeros en adoptar el esquema de búsqueda utilizando índices para realizar seguimiento de audio, las pruebas se realizaron con

grabaciones en las que únicamente existe un instrumento (piano) y que fueron efectuadas en un ambiente muy controlado. Sin ruidos y otros tipos de degradaciones. Trabajos posteriores podrían incorporar el uso de interpretaciones que incluyan:

- Instrumentos diferentes al piano.
- Varios instrumentos a la vez y/o voz.
- Grabaciones con degradaciones como ruido, ecualizaciones, cambios en el volumen, cuantizaciones etc.
- Música en vivo.

2. En este trabajo se hizo uso de una huella de audio basada en la entropía y un índice basado en LSH. Otros esquemas podrían optar por utilizar otras huellas de audio que muestren mayor consistencia cuando se trate con interpretaciones diferentes. Adicionalmente, se utilizó el índice LSH debido a que la firma utilizada realiza una codificación de los vectores de características a 0's y 1's. Si se cambia la huella utilizada probablemente sea conveniente utilizar un índice diferente.

3. Una característica muy evidente del índice LSH es que es altamente paralelizable. Una extensión al trabajo presentado consistiría en paralelizar el proceso de búsqueda en el índice de manera que las instancias creadas se procesen en paralelo. Ésto disminuiría el tiempo de ejecución del algoritmo en general mejorando así su capacidad de desempeñarse en tiempo real. Para ello se podría utilizar CUDA, una plataforma de cómputo en paralelo creada por NVIDIA.

4. Implementar una interfaz para pruebas en tiempo real, que obtenga el audio del ambiente y utilice el algoritmo presentado para realizar el alineamiento.

5. La implementación del alineamiento de audio presentada en este trabajo se concibió para que pudiera ser utilizada en aplicaciones de tiempo real. Sin embargo, si la ejecución en tiempo real no es un objetivo, se podría utilizar una ligera variación de éste trabajo para realizar alineamiento de interpretaciones de manera *off-line*. Durante algunos experimentos realizados se comprobó que modificando los parámetros de la

huella para la extracción de características de manera que ésta tenga mayor definición, los resultados del alineamiento mejoraban considerablemente. Utilizando marcos con un translape de aproximadamente 3 ms se obtiene una firma de audio lo suficientemente definida como para realizar un seguimiento muy preciso. Desafortunadamente, utilizar un translape de 3 ms simplemente no es viable en un contexto de tiempo real.

Apéndice A

Tabla de Interacción de Parámetros

Parámetro	Acciones		Compensación
k	+	Aumenta la posibilidad de encontrar un candidato que cumpla las condiciones. Se pueden utilizar posiciones cuyas matrices no se parecen a la buscada	Disminuir $windowMultiplier$
	−	Disminuye la posibilidad de encontrar un candidato adecuado. Las posiciones candidato tienen matrices muy parecidas a la buscada	Aumentar $windowMultiplier$
nB	+	Aumenta el tiempo de comparación entre matrices. Si se utilizan bandas sin contenido útil se puede afectar el resultado de la comparación.	Utilizar una distancia que no tome en cuenta inserciones y borrados como Hamming.
	−	Disminuye el tiempo de comparación. Si se utilizan menos bandas de las necesarias no se toma en cuenta información importante	Usar una distancia con inserciones y borrados como Levenshtein.
$windowSize$	+	Disminuye la "fineza" del seguimiento. Aumenta el tiempo disponible para realizar el procesamiento	
	−	Aumenta la "fineza" del seguimiento. Disminuye el tiempo disponible para realizar el procesamiento	
$windowMultiplier$	+	Permite "saltos" más grandes. Poco probable que el algoritmo decida "mantenerse".	Disminuir k. Utilizar una política de elección que tome en cuenta la cercanía en tiempo.
	−	Permite "saltos" pequeños. Si existen secciones faltantes el seguimiento podría ser inadecuado.	Aumentar k para aumentar las posibilidades de avanzar.
$howManyMaps$	+	Aumenta la cantidad de *buckets* obtenidos por vector.	Aumentar hmb y/o disminuir bV.
	−	Disminuye la cantidad de *buckets* obtenidos por vector	Aumentar bV y/o disminuir hmb.
$howManyBits$	+	Disminuye la "densidad" de los *buckets* y ∴ la cantidad de posiciones a revisar.	Aumentar $howManyMaps$ y/o $bitVariations$.
	−	Aumenta la "densidad" de los *buckets* y ∴ la cantidad de posiciones a revisar.	Disminuir $howManyMaps$ y/o $bitVariations$.
$bitVariations$	+	Aumenta las variaciones creadas por cada vector y ∴ la cantidad de *buckets* obtenidos.	Aumentar hmb y/o disminuir hmm.
	−	Disminuye las variaciones creadas por cada vector y ∴ la cantidad de *buckets* obtenidos.	Aumentar hmm y/o disminuir hmb.

Tabla A.1: Interacción entre parámetros.

Apéndice B

Partituras

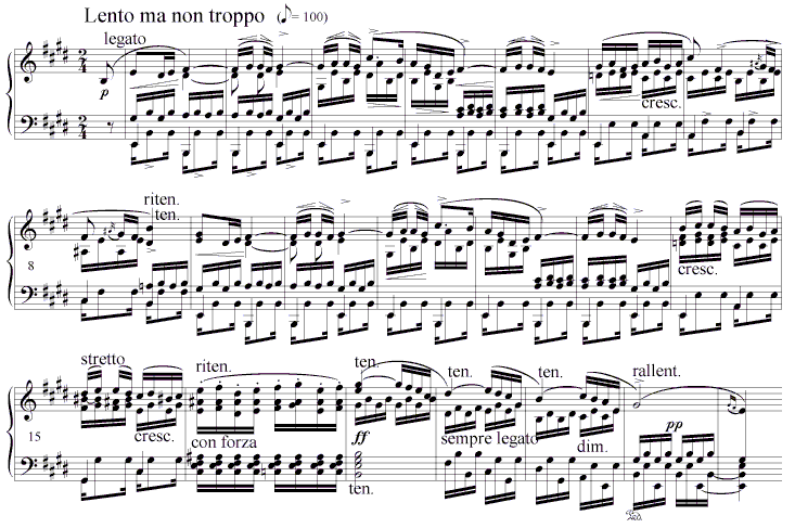


Figura B.1: Etude en E Major, Op. 10, no. 3, barras 1-21, Frédéric Chopin.

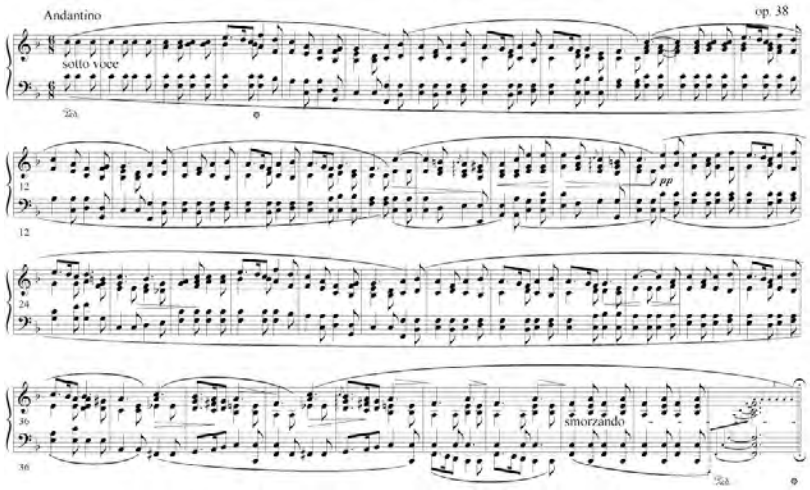


Figura B.2: Ballade Op. 38, barras 1-45, Frédéric Chopin.

Apéndice C

Formas de Onda

Únicamente se incluyen las formas de onda de las grabaciones utilizadas en los ejemplos presentados en las secciones **??** y **??**. Todas las grabaciones pueden ser obtenidas en línea en **?**.

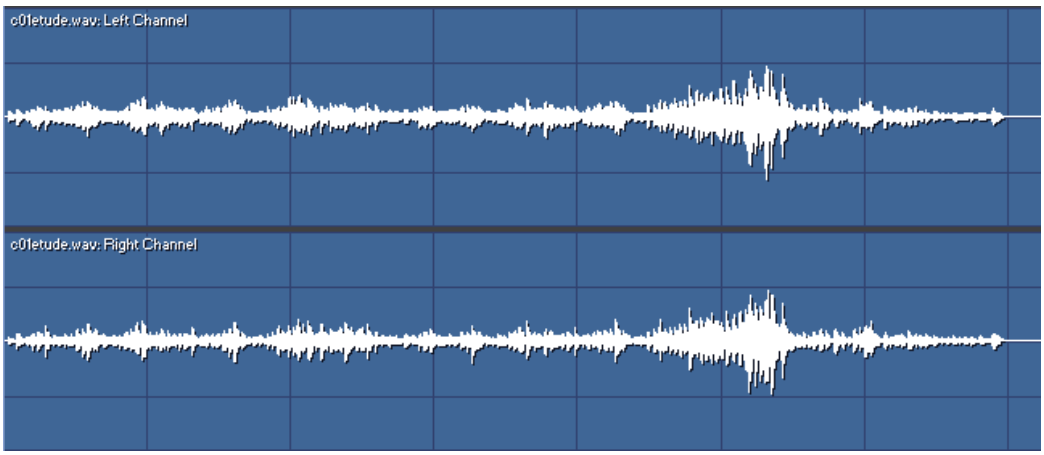


Figura C.1: Forma de onda Etude #1, duración 1:26:07

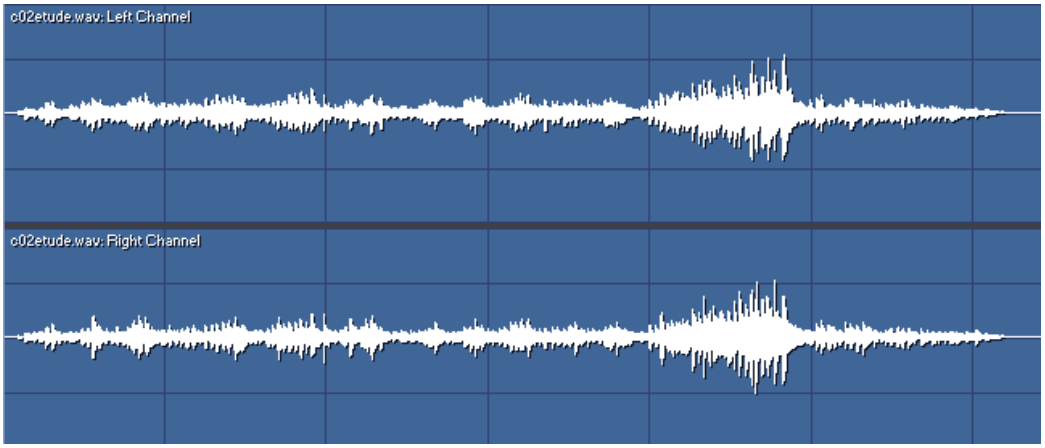


Figura C.2: Forma de onda Etude #2, duración 1:16:46

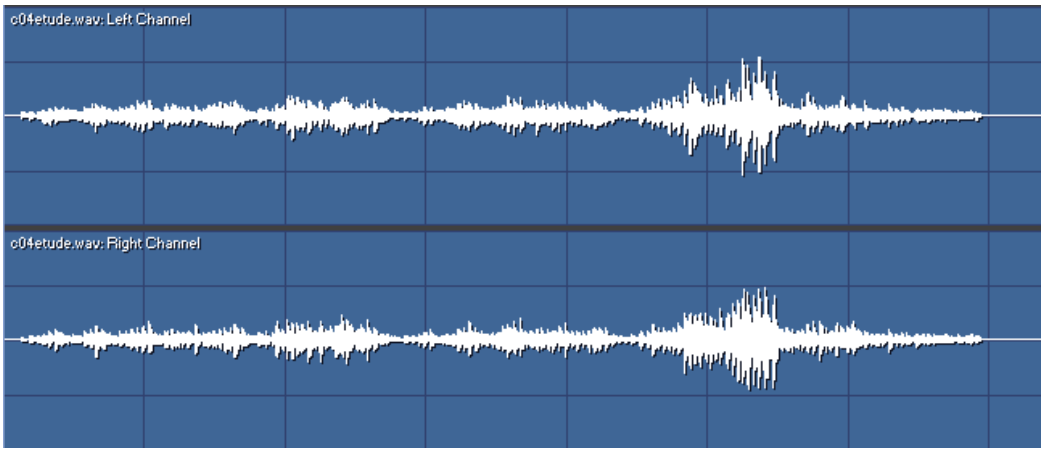


Figura C.3: Forma de onda Etude #4, duración 1:27:72

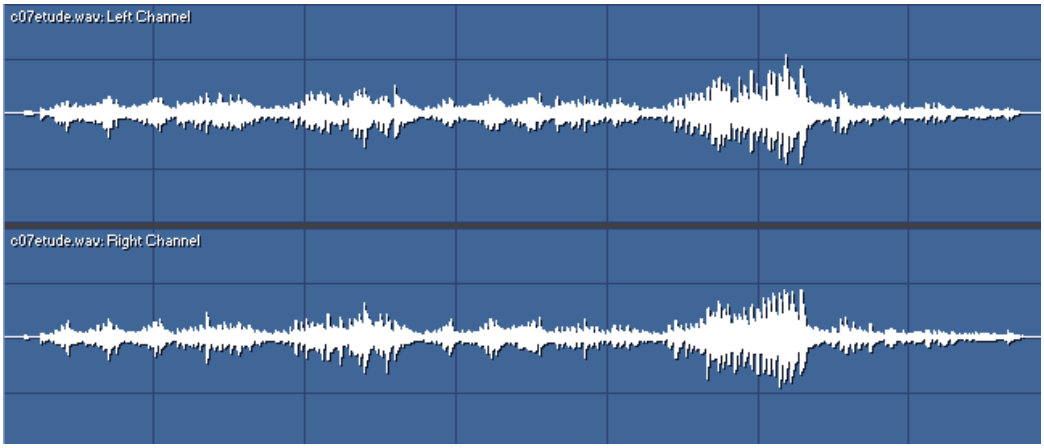


Figura C.4: Forma de onda Etude #7, duración 1:21:84

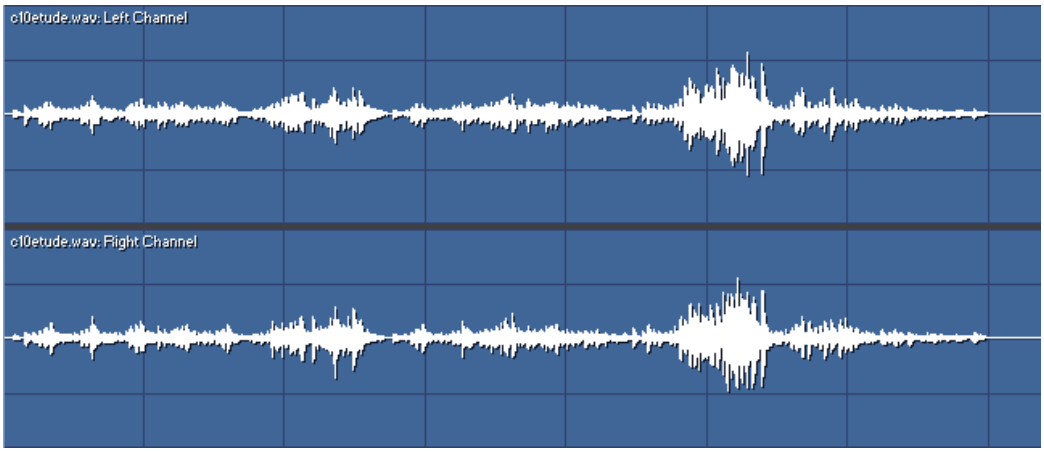


Figura C.5: Forma de onda Etude #10, duración 1:27:80

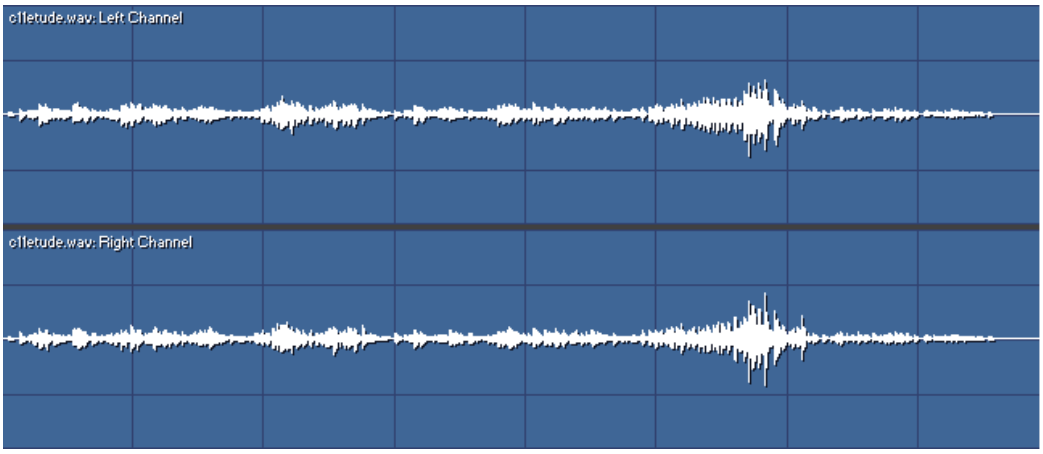


Figura C.6: Forma de onda Etude #11, duración 1:34:38

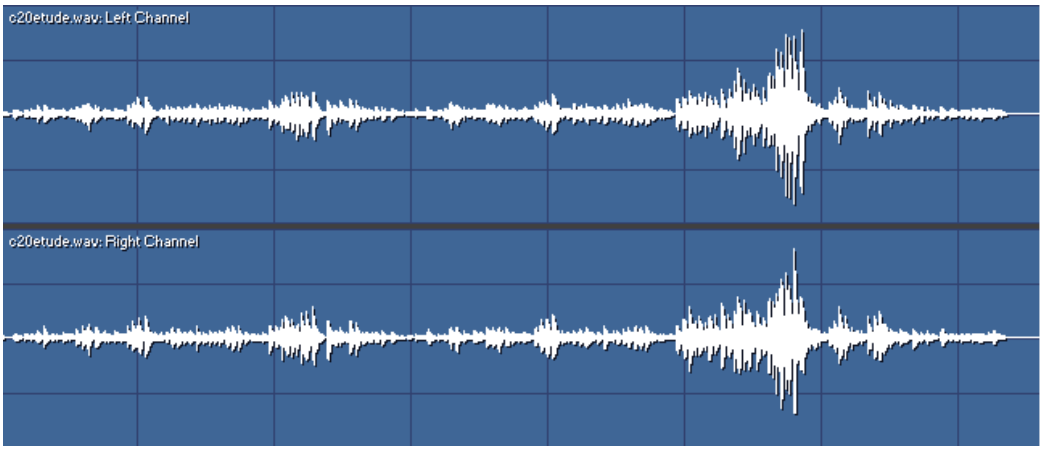


Figura C.7: Forma de onda Etude #20, duración 1:30:36

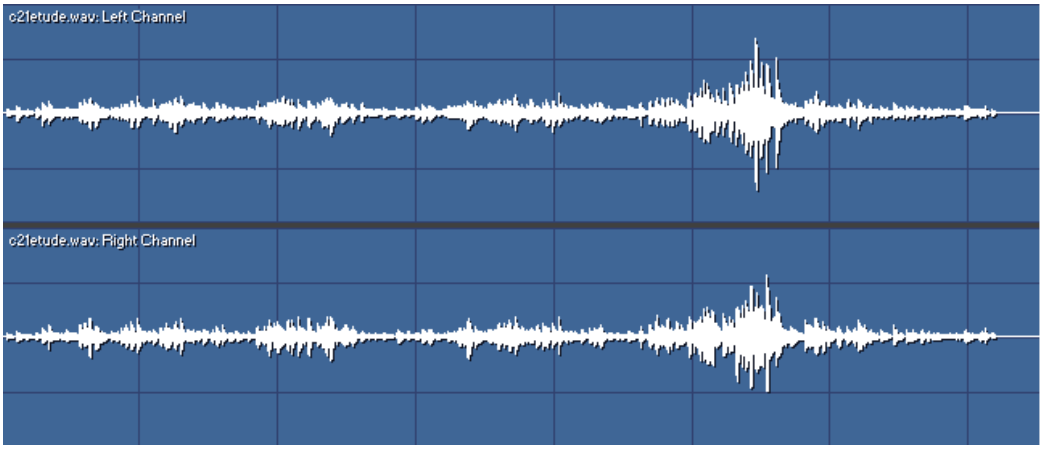


Figura C.8: Forma de onda Etude #21, duración 1:29:47

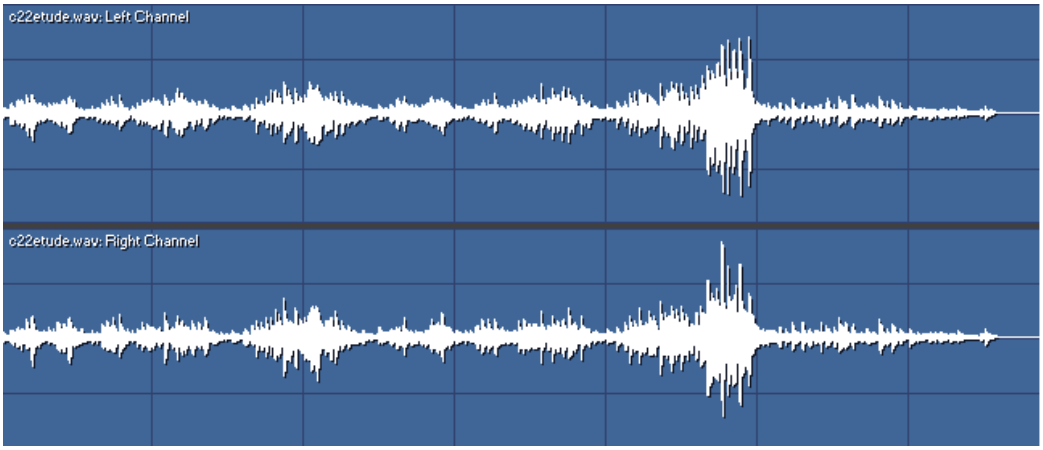


Figura C.9: Forma de onda Etude #22, duración 1:21:74

Referencias

Andoni, A., y Indyk, P. (2008). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications ACM*, *51*, 117–122.

Bellman, R. (1961). *Adaptive control processes - A guided tour.*. Princeton University Press.

Camarena, A., y Chávez, E. (2006). A robust entropy-based audio-fingerprint. *IEEE International Conference on Multimedia and Expo*, *1*, 1729–1731.

Cannam, C., Landone, C., y Sandler, M. (2010). Sonic visualiser: An open source application for viewing, analysing, and annotating music audio files. *Proceedings of the ACM Multimedia 2010 International Conference*, (pp. 1467–1468).

Cano, P., Loscos, A., y Bonada, J. (1999). Score-performance matching using hmms. *Proceedings of International Computer Music Conference (ICMC)*.

Chou, T., Chen, W., Wang, S., Chang, K., y Chen, H. (2012). Real-time polyphonic score following system. *IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, (pp. 205–210).

Cont, A. (2006). Realtime audio to score alignment for polyphonic music instruments using sparse non-negative constraints and hierarchical hmms. *IEEE International Conference on Acoustics and Speech Signal Processing (ICASSP)*.

Cont, A. (2010). A coupled duration-focused architecture for realtime music to score alignment. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, *32(6)*, 974–987.

Dannenberg, R. (1984). An on-line algorithm for real-time accompaniment. *Proceedings of the International Computer Music Conference (ICMC)*, (pp. 193–198).

Dannenberg, R., y Grubb, L. (1989). Real-time scheduling and computer accompaniment. *MIT Press Series in System Development Foundation Benchmark*, (pp. 225–261).

Davis, S. B., y Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *28*, 357–366.

de Cheveigné, A. (2006). Múltiple f0 estimation. *Computational Auditory Scene Analysis: Principles, Algorithms and Applications*.

Desain, P., Honing, H., y Heijink, H. (1997). Robust score-performance matching: Taking advantage of structural information. *Proceedings of International Computer Music Conference (ICMC)*.

Deza, E., y Deza, M. M. (2009). *Encyclopedia of Distances*. Springer.

Dixon, S. (2005). Live tracking of musical performances using on-line time warping. *8th International Conference on Digital Audio Effects*.

Dixon, S. (2010). Simon dixon's personal page. Online. Http://www.eecs.qmul.ac.uk/ simond/match/.

Dixon, S., y Widmer, G. (2005). Match: A music alignment tool chest. *6th International Conference on Music Information Retrieval*, (pp. 492–497).

Fine, S., Singer, Y., y Tishby, N. (1998). The hierarchical hidden markov model : Analysis and applications. *Machine Learning*, *32*(1), 41–62.

Fujishima, T. (1999). Realtime chord recognition of musical sound: A system using common lisp music. *Proceedings of the International Computer Music Conference*.

Gerhard, D. (2003). *Pitch extraction and fundamental frequency: History and current techniques*. University of Regina.

Gionis, A., Indyk, P., y Motwani, R. (1999). Similarity search in high dimensions via hashing. *Proceedings of the 25th International Conference on Very Large Databases*, *1*, 518–529.

Goebl, W. (2001). Melody lead in piano performance: Expressive device or artifact? *Journal of the Acoustical Society of America*, *110*(1), 563–572.

Goebl, W. (2013). Werner goebl's personal page. Online. Http://iwk.mdw.ac.at/goebl/mp3.html.

Grubb, L., y Dannenberg, R. (1998). Enhanced vocal performance tracking using multiple information sources. *Proceedings of ICMC*, (pp. 37–44).

Haitsma, S., y Kalker, T. (2002). A highly robust audio fingerprintig system. *ISMIR*.

Haitsma, S., Kalker, T., y Oostveen, J. (2001). Robust audio hashing for content identification. *Content Based Multimedia Indexing*.

Hamming, R. (1950). Error detecting and error correcting codes. *Bell System Technical Journal*, *29*, 147–160.

Hirschber, D. S. (1975). A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, *18*, 341–343.

Hu, N., Dannenberg, R., y Tzanetakis, G. (2003a). Polyphonic audio matching and alignment for music retrieval. *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*.

Hu, N., Dannenberg, R., y Tzanetakis, G. (2003b). Polyphonic audio matching for score following and intelligent audio editors. *Proceedings of the ICMC*.

Johnston, J. D. (1988). Transform coding of audio signals using perceptual noise criteria. *IEEE Journal on Selected Areas in Communications*, *6*(2), 314–332.

Jordan, M. I. (1998). *Learning in Graphical Models*. MIT Press.

Jordanus, A. (2007). *Score Following: An Articially Intelligent Musical Accompanist*. Tesis de Maestría, University of Edinburgh.

Kirchhoff, H., y Lerch, A. (2011). Evaluation of features for audio-to-audio alignment. *Journal of New Music Research*, *40*(1), 27–41.

Krause, E. F. (1986). *Taxicab Geometry: An Adventure in Non-Euclidean Geometry*. Courier Dover Publications.

Levenshtein, V. (1966). Binary codes capable of correction deletions, insertions and reversals. *Soviet Physics Doklady*, *10*, 707–710.

Okanohara, D., y Sadakane, K. (2006). Practical entropy-compressed rank/select dictionary. *Computing Research Repository*.

Orio, N., y Déchelle, F. (2001). Score following using spectral analysis and hidden markov models. *Proceedings of International Computer Music Conference (ICMC)*.

Orio, N., Lemouton, S., y Schwarz, D. (2003). Score following: State of the art and new developments. *Proceedings of the Conference on New Interfaces for Musical Expression (NIME)*.

Orio, N., y Schwarz, D. (2001). Alignment of monophonic and polyphonic music to a score. *Proceedings of International Computer Music Conference (ICMC)*.

Pohlmann, K. (1995). *Principles of Digital Audio*. McGraw-Hill.

Puckette, M., y Lippe, C. (1992). Score following in practice. *Proceedings of International Computer Music Conference (ICMC)*.

Rabiner, L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*, 257–286.

Rabiner, L. R., y Juang, B. H. (1993). *Fundamentals of Speech Recognition*. Prentice Hall.

Raphael, C. (1999). Automatic segmentation of acoustic musical signals using hidden markov models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *21(4)*, 360–370.

Raphael, C. (2006). Aligning music audio with symbolic scores using a hybrid graphical model. *Machine Learning*, *65*(2-3), 389–409.

Santoyo, F., y Chávez, E. (2012). *Índices Comprimidos para Búsqueda por Proximidad en Series de Tiempo*. Tesis de Maestría, Universidad Michoacana de San Nicolás de Hidalgo.

Santoyo, F., Chávez, E., y Tellez, E. (2012). Compressing locality-sensitive hashing tables.

vanKasteren, T. (2006). *Realtime Tempo Tracking using Kalman Filtering*. Tesis de Maestría, University of Amsterdam.

Vercoe, B. (1984). The synthetic performer in the context of live musical performance. *Proceedings of the International Computer Music Conference (ICMC)*.

Wold, E., Blum, T., Keislar, D., y Wheaton, J. (1996). Content-based classification, search and retrieval of audio. *IEEE Multimedia*, *3*, 27–36.