Similitud melódica como transformación de cadenas - I

Paco Gómez

He aquí una nueva serie de tres artículos que investigan desde un punto de vista divulgativo la relación entre la similitud melódica y la teoría de cadenas (una rama de la computación). La similitud melódica, bajo ciertas hipótesis que simplifican el complejo fenómeno que es, se puede concebir como un problema de comparación de cadenas, donde las cadenas aquí representan sucesiones de notas. En un artículo titulado *Comparison of musical sequences* Mongeau y Sankoff [MS90] explotaron esta idea. En esta serie glosaremos su trabajo con detalle. En el artículo de este mes discutimos el concepto de similitud melódica, por la parte musical, el concepto de comparación de cadenas, con especial énfasis en la distancia de edición, y finalmente analizamos los aspectos computacionales de dicha distancia. En el siguiente artículo veremos cómo Mongeau y Sankoff adaptaron la distancia de edición para computar la disimilitud entre dos melodías dadas. En el último artículo de la serie examinaremos los experimentos llevados a cabo por estos autores para comprobar la bondad de su medida. Esos experimentos consisten en medir la similitud melódica de las nueve variaciones sobre el tema *Ah, vous dirai-je, Maman*, K. 265, de Mozart y analizar los resultados obtenidos.

1 Similitud melódica

Los fenómenos musicales son básicamente perceptuales y cognitivos en su naturaleza ([Deu98], página 158). Esta aseveración puede sonar a los oídos de hoy como evidente, pero la tradición racionalista del análisis musical la relegó al olvido durante mucho tiempo. De hecho, la psicología misma empezó a reconocer la música como objeto de estudio serio hace solo unas pocas décadas. Hasta entonces el análisis musical se había basado en las matemáticas (desde la tradición pitagórica al sistema de doce tonos de Schoenberg), la física (empezando con el trabajo de Helmhotz [Hel85]), y la música (análisis schenkeriano, análisis armónico, etc.), y los aspectos cognitivos de la música se habían desatendido por completo.

La melodía se encuentra entre los fenómenos musicales más investigados. Los teóricos de la música han examinado miles de melodías con el fin de describir sus características estructurales, mientras que los psicólogos han ceñido su atención a la manera en que el ser humano percibe y responde a la melodía. Un teórico de la música bien puede preguntarse cuáles son los elementos constituyentes de la melodía, y su respuesta, probablemente, vendrá dada en términos de altura y relaciones de duración, tales como dirección melódica, relación interválica, altura más grave y más aguda, entre otras [Ort37]; o incluso en términos de atributos más abstractos, tales como propincuidad, repetición de tonos y finalidad [Lun67]. En contraste con esto, un psicólogo de la música se podría preguntar, por ejemplo, cuáles son los factores psicológicos que transforman una sucesión de notas en una melodía, y esta vez, probablemente, la respuesta se presentaría en términos de leyes de percepción (Bower y Hilgard [BH81]), esquemas melódicos (Dowling and Hardwood [DH86]), o modelos de estructuración jerárquica perceptual (West et al. [WHC85], Sloboda [Slo85], Lerdahl [LJ83]). Hoy en día es inconcebible estudiar el fenómeno melódico sin examinar ambas facetas.

Pero cuando se trata de procesar música o de establecer algún esquema cuantitativo o en última instancia computacional, ¿es posible incorporar el conocimiento proveniente de ambos campos? Cuando empezó el estudio computacional de la música, los modelos de entonces no permitían tener esos aspectos en consideración. Aunque hoy los pasos son más seguros en esa dirección, todavía son lentos. En este artículo vamos a estudiar un modelo computacional de similitud melódica. El modelo simplifica en buena medida la complejidad de ese fascinante fenómeno que es la melodía. A cambio obtiene una medida que permite comparar melodías bajo ciertas condiciones. Sigue, qué duda cabe, una evaluación del método para ver cuánto afectó la mencionada simplificación.

La similitud melódica es un concepto fundamental, tanto desde el punto de vista teórico como desde el práctico. Es esencial en el proceso musical porque sirve como evaluación de la variación del material, del reconocimiento del estilo y del compositor, de la interpretación, del aprendizaje musical, o en la clasificación musical; véase McAdams and Matzkin [MM01]. Acicateados por el creciente número de aplicaciones de la similitud melódica (sistemas de recomendación, leyes de propiedad intelectual, búsqueda en bases de datos musicales, clasificación de estilos, atribución de obras dudosas), muchos matemáticos e informáticos han diseñado algoritmos para computar la similitud melódica basados en muy diversas técnicas: medidas geométricas, distancias de transporte, medidas de probabilidad, similitud estadística, proximidad geográfica, entre otros; para una lista exhaustiva, véase [HSF98] y las referencias allí listadas.

Mongeau y Sankoff, en su artículo del año 90 [MS90], arguyen que los aspectos puramente musicales son bastante difíciles de evaluar y que están sujetos a un alto grado de subjetividad (en realidad, van más lejos y dicen *arbitrariedad*, que es una palabra con implicaciones más serias). Deciden centrarse en dos variables fundamentales: la altura del sonido y las duraciones (el ritmo). En su trabajo adaptan las distancias de edición clásicas (la distancia de Levenshtein) y la amplía con más operaciones. Ello les permite definir una distancia con más poder de discriminación y aplicar a la comparación de piezas musicales.

2 Distancias entre cadenas de símbolos

En Informática el problema de la distancia entre cadenas es un problema que aparece en muchos contextos y que tiene muchas aplicaciones. En su versión más básica se formula como sigue: dadas dos cadenas A, B de símbolos (tomados de un alfabeto común), hallar el mínimo número de operaciones que hay que realizar para transforma A en B. Las operaciones se definen previamente y las tres más elementales son *borrado*, *inserción* y *sustitución*. Ese número mínimo de operaciones se toma como la distancia entre A y B. Como veremos en las aplicaciones prácticas se añaden operaciones más complejas tales como *fragmentación* y *consolidación*.

Esta distancia fue descubierta por Levenshtein en el año 1965 y publicada en una revista rusa de informática. Se la conoce como *distancia de Levenshtein* y también como *distancia de edición*. A partir de ella se han definido otras muchas: distancias de Levenshtein ponderadas, distancia de Damerau-Levenshtein (que permite trasposiciones), distancias de Hamming, entre otras.

Antes de definir formalmente la distancia de edición, vamos a poner un ejemplo. Las operaciones usadas serán las que mencionamos arriba: borrado, inserción y sustitución. Asignaremos un coste de 1 a cada operación. Consideremos la cadena $A = TENER$ y $B = PERDER$. Una posible secuencia de operaciones para transformar A en B es la siguiente:

1. Borrado de T: $TENER \Longrightarrow ENER$.

2. Inserción de P: $ENER \Longrightarrow PENER$.

3. Sustitución de N por R: $PENER \Longrightarrow PERER$.

4. Inserción de D: $PERER \Longrightarrow PERDER$.

La transformación se ha hecho en 4 operaciones. No es la mínima ya que las operaciones 1) y 2) se pueden reemplazar por una sustitución directa a coste 1 solo. En ese caso, el número de operaciones sería mínimo y la distancia valdría 3.

Para definir formalmente la edición hace falta especificar un conjunto de operaciones y a cada una de ellas asignarles un coste. Una vez hecho eso, la distancia de edición es, como dijimos, el número mínimo de operaciones necesario para transformar una cadena en otra.

2.1 Algoritmo para calcular la distancia de edición

En esta sección vamos a estudiar los aspectos computacionales de la distancia de edición. Su definición nos parece clara, pero ¿cómo es posible calcularla? Sean A, B dos cadenas de longitudes n y m, respectivamente, con caracteres pertenecientes a un alfabeto común. Un algoritmo clásico para resolver el problema de la distancia de edición es la *programación dinámica*. Esta técnica algorítmica se aplica a problemas en que la solución local es parte de la solución global, como ocurre en el caso que nos ocupa.

Por comodidad en la descripción del algoritmo, designamos por $A[1..i]$ la subcadena $(a_1, \ldots, a_i)$ de A, donde $1 \leq i \leq n$; y análogamente con la subcadena $B[1..j]$ de B, con $1 \leq j \leq m$. El algoritmo calcula la distancia de A a B usando una matriz como estructura de datos, matriz que tiene dimensiones $(n+1) \times (m+1)$. Dicha matriz sirve para almacenar las distancias entre todas las subcadenas $A[1..i]$ y $B[1..j]$. En un paso genérico el algoritmo calcula la distancia $d(A[1..i], B[1..j])$ y para ello se apoya en las distancias de subcadenas más pequeñas, en particular, en las distancias entres subcadenas. Si c_I, c_B, c_S son, respectivamente, los coste de la inserción, borrado y sustitución, la distancia $d(A[1..i], B[1..j])$ se calcula con la fórmula siguiente:

$$d(a_i, b_j) = \text{mín} \begin{cases} d(a_{i-1}, b_j) + c_I, \\ d(a_i, b_{j-1}) + c_B, \\ d(a_{i-1}, b_{j-1}) + c_S \end{cases}$$

Es, como se puede apreciar, una fórmula que usa los valores previos para calcular el valor actual. La solución se construye de manera local, en cada paso, y la solución global es el resultado del procesamiento de todos las subcadenas de A y B. No obstante, dado que en cada paso se usan valores de subcadenas más pequeñas, solo se procesan dos caracteres, a_i y b_j, en cada paso del algoritmo. Para que la sucesión de operaciones no se reduzca a las sustituciones, en las aplicaciones suele aparecer la condición $c_S < c_I + c_B$.

Como paso de inicialización el algoritmo necesita tener calculados las distancias de $d(A[1..i], \emptyset)$ y $d(\emptyset, B[1..j])$, donde $i = 1, \ldots, n$, $j = 1, \ldots, m$ y $\emptyset$ es la cadena vacía. El valor de $d(A[1..i], \emptyset)$ es $j \cdot c_B$ y el de $d(\emptyset, B[1..j])$ es $i \cdot c_I$.

A continuación se muestra un pseudocódigo (adaptado de [Wik13]).

```
int LevenshteinDistance(char cad1[1..longCad1], char cad2[1..longCad2])
   // d es una matriz con  longCad1+1 filas y longCad2+1 columnas
   declare int d[0..longCad1, 0..longCad2]
   // i y j  se usan como variables para el bucle iterativo
   declare int i, j, costeBorrado, costeInsercion, costeSustitucion

(1)   for i from 0 to longCad1
       d[i, 0] := i*costeInsercion
(2)   for j from 0 to longCad2
       d[0, j] := j*costeBorrado

(3)   for i from 1 to longCad1
(4)      for j from 1 to longCad2
(5)           if cad1[i] = cad2[j] then d[i, j]:=d[i-1, j-1]
               else
                    d[i, j] := minimo(
                                d[i-1, j] + costeBorrado,        // Borrado
                                d[i, j-1] + costeInsercion,      // Inserción
                                d[i-1, j-1] + costeSustitucion   // Sustitución
                           )

   return d[longCad1, longCad2]
```

La prueba de corrección del algoritmo se basa en un invariante que se mantiene a lo largo de todo el algoritmo: el hecho de que el elemento $d(i, j)$ de la matriz contiene la distancia de edición de las subcadenas $A[1..i]$ y $B[1..j]$. No es inmediatamente evidente que $d(i, j)$ proporciona de hecho el número mínimo de transformaciones entre dichas subcadenas; require una prueba por inducción que no damos aquí. En ciertos contextos, no solo es necesario la distancia entre las cadenas, sino también la sucesión de operaciones que transforma una en la otra. En un ejemplo desarrollado más abajo se muestra cómo conseguir esa sucesión.

2.2 Un ejemplo de la distancia de Levenshtein

Tomemos para nuestro ejemplo dos inocentes cadenas, dicho en el sentido estricto de la palabra, $A = \{político\}$ y $B = \{corrupción\}$. En un mundo ideal, volterianamente cándido, la distancia entre ambas cadenas debería ser infinita. En el mundo que nos ocupa, y más en el de las secas cadenas de caracteres, sabemos que esa distancia es finita, dolorosamente finita. Computemos cuán finita. Nos ayudaremos del *applet* que ha escrito Scott Fescher [Fes13] para ilustrar el funcionamiento de la distancia de edición; las figuras de más abajo han sido generadas con ayuda de ese *applet*. Como pesos para las operaciones tomaremos $c_I = c_B = c_S = 1$, esto es, la distancia original de Levenshtein. Obsérvese que se cumple la condición $c_S < c_I + c_B$.

El primer paso, como sabemos, es la inicialización. Esta consiste en calcular las distancias de la cadena vacía $\{\emptyset\}$ a las cadenas $A = \{político\}$ y $B = \{corrupción\}$; véanse los bucles (1) y (2) del pseudocódigo presentado más arriba. Todas las operaciones se reducen a inserciones (en la fila) y a borrados (en la columna), como se puede apreciar en la matriz de la figura 1.

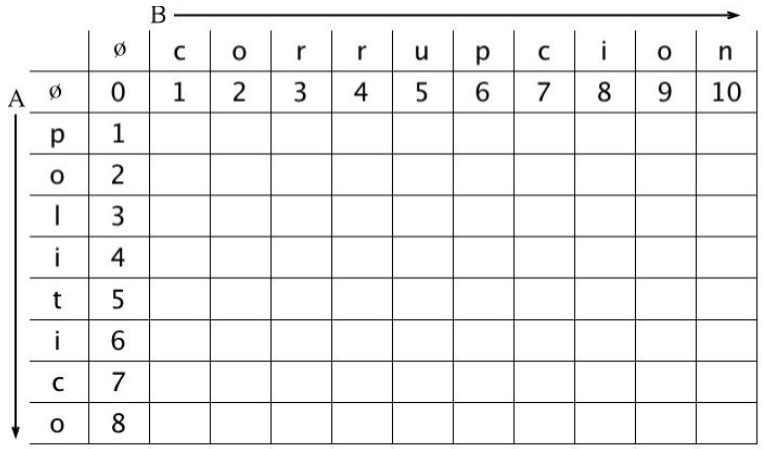

Figura 1: Inicialización del algoritmo de la distancia de edición.

A continuación empiezan a procesarse el primer carácter de A y todas las subcadenas $B[1..j]$ de B para $j = 1, \ldots, 10$ (10 es la longitud de la $\{corrupción\}$). En la figura 2 nos hemos parado en el cálculo de la distancia de las subcadenas subcadenas $\{p\}$ y $\{corrup\}$. En ese momento el algoritmo tiene que rellenar el elemento $(2, 7)$ de la matriz. Primero se comprueba que la los caracteres a procesar son distintos. No es el caso aquí, pues ambos se reducen al carácter $\{p\}$. La distancia se iguala a la del elemento $(1, 6)$, cuyo valor es 5.

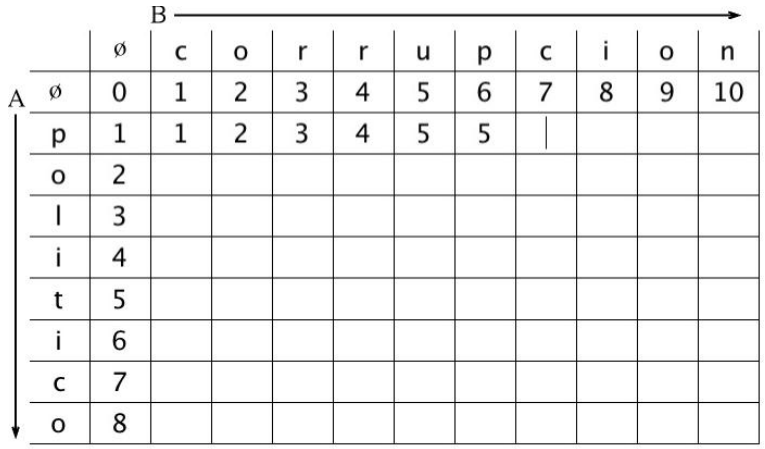

Figura 2: Detalle del cálculo de la distancia de las subcadenas $\{p\}$ y $\{corrup\}$.

En la figura 3 vemos unos cuantos casos más donde se dan la igualdad entre caracteres de las dos cadenas (aparecen rodeados con un círculo rojo). Véase el *if*, línea (5), en el pseudocódigo arriba.

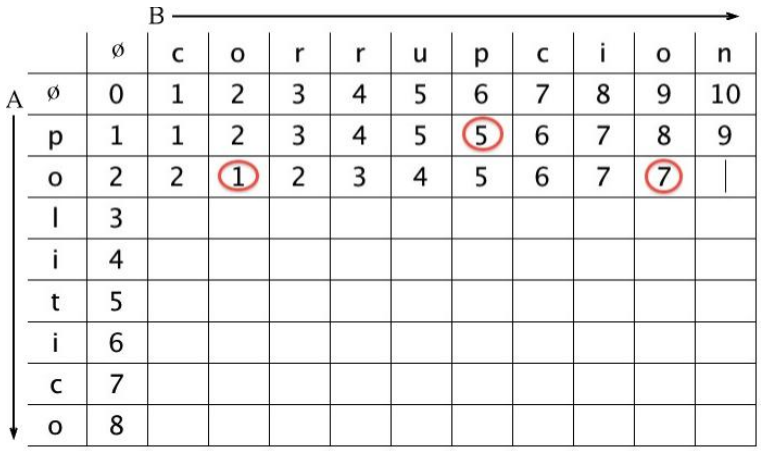

Figura 3: Casos en que los caracteres a procesar son iguales.

En un paso genérico, cuando los caracteres a procesar no son iguales, el algoritmo calcula las distancias a partir de tres distancias inmediatamente anteriores. En el caso de la figura 4 se va a calcular la distancia $d(6,5)$. Las distancias de los anteriores elementos son $d(6,4) = 4, d(5,4) = 3$ y $d(5,5) = 3$. El algoritmo especifica que la nueva distancia ha de ser el mínimo entre los números $\{d(6,4)+1, d(5,4)+1, d(5,5)+1\}$, que es 4.

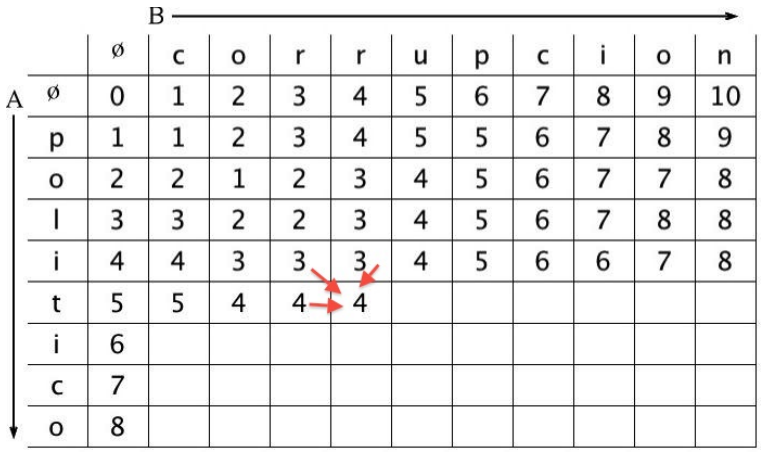

Figura 4: Paso genérico del algoritmo.

Si continuamos todo el proceso hasta el final, la matriz que obtenemos es la de la figura 5. La distancia de edición viene dada por el elemento $(9, 11)$, el más abajo a la izquierda, rodeado por un círculo rojo. Su valor es 7.

		B										
		ø	c	o	r	r	u	p	c	i	o	n
A	ø	0	1	2	3	4	5	6	7	8	9	10
	p	1	1	2	3	4	5	5	6	7	8	9
	o	2	2	1	2	3	4	5	6	7	7	8
	l	3	3	2	2	3	4	5	6	7	8	8
	i	4	4	3	3	3	4	5	6	6	7	8
	t	5	5	4	4	4	4	5	6	7	7	8
	i	6	6	5	5	5	5	5	6	6	7	8
	c	7	6	6	6	6	6	6	5	6	7	8
	o	8	7	6	7	7	7	7	6	6	6	7

Figura 5: Matriz completa de la distancia de edición.

El algoritmo se puede ampliar con un sistema de punteros de tal manera que se puede reconstruir la sucesión de operaciones que transforma una cadena en la otra. Dada una distancia $d(i, j)$ a calcular en un paso genérico del algoritmo, si el mínimo se alcanza con la distancia $d(i, j-1)$ estamos ante una inserción, si se alcanza con $d(i-1, j)$ estamos ante un borrado y si se alcanza con $d(i-1, j-1)$, ante una sustitución. A partir de la distancia final, y disponiendo de esta información, se puede recorrer la matriz desde el elemento $(n+1, m+1)$ hasta el $(1, 1)$ y listar la sucesión de operaciones. En general, el camino que va del elemento que da la distancia de edición hasta el $(1, 1)$ no es único. En la figura 6 se muestra un posible camino. Los números rodeados por un círculo rojo corresponden a los momento en que los caracteres a procesar eran idénticos.

		B										
		ø	c	o	r	r	u	p	c	i	o	n
A	ø	0	1	2	3	4	5	6	7	8	9	10
	p	1	1	2	3	4	5	5	6	7	8	9
	o	2	2	1	2	3	4	5	6	7	7	8
	l	3	3	2	2	3	4	5	6	7	8	8
	i	4	4	3	3	3	4	5	6	6	7	8
	t	5	5	4	4	4	4	5	6	7	7	8
	i	6	6	5	5	5	5	5	6	6	7	8
	c	7	6	6	6	6	6	6	5	6	7	8
	o	8	7	6	7	7	7	7	6	6	6	7

Figura 6: Obtención de la sucesión de operaciones.

Si I y S representan inserción y borrado, la sucesión de operaciones de la figura anterior es (S, S, S, S, S, I, I) y la sucesión de transformaciones es:

$$politico \Longrightarrow Colitico \Longrightarrow CORitico \Longrightarrow CORRtico \Longrightarrow CORRUico$$

$$\Longrightarrow CORRUPco \Longrightarrow CORRUPCIo \Longrightarrow CORRUPCION$$

3 Conclusiones

Como vemos la distancia entre político y corrupción no es mucha, siete pasos, aunque, como dijimos antes, debería ser infinita o al menos ser finita solo en el mundo de la computación. El mes que viene veremos cómo se puede aplicar la distancia de edición a la comparación de cadenas con significado musical.

Bibliografía

[BH81] G. H. Bower and E. R. Hilgard. *Theories of learning*. Prentice-Hall, Englewwod Cliffs, NJ, 1981.

[Deu98] D. Deutsch. *The Psychology of Music*. Academic Press, 1998.

[DH86] W. J. Dowling and D. L. Hardwood. *Music cognition*. Academic Press, Orlando, FL, 1986.

[Fes13] S. Fescher. Edit Distance ILM. http://csilm.usu.edu/lms/nav/activity.jsp?sid=__shared&cid=emready@cs5070_projects&lid=10, consultado en octubre de 2013.

[Hel85] H. Von Helmholtz. *On the sensations of tone as a physiological basis for the theory of music*. Dover, New York, 1954 (publicado originalmente en 1885).

[HSF98] W. B. Hewlett and E. Selfridge-Field. *Melodic Similarity: Concepts, Procedures, and Applications*. MIT Press, Cambridge, Massachussetts, 1998.

[LJ83] F. Lerdahl and R. Jackendoff. *A Generative Theory of Tonal Music*. MIT Press, Cambridge, Massachussetts, 1983.

[Lun67] R. W. Lundin. *An objective psychology of music*. Ronald Press (segunda edición), New York, 1967.

[MM01] S. McAdams and D. Matzkin. Similarity, invariance and musical variation. *Annals of the New York Academy of Sciences*, 90:62–76, 2001.

[MS90] M. Mongeau and D. Sankoff. Comparison of musical sequences. *Computers and the Humanities*, 24:161–175, 1990.

[Ort37] O. Ortmann. Interval frequency as a determinant of melodic style. *Peabody Bulletin*, pages 3–10, 1937.

[Slo85] J. A. Sloboda. *The musical mind*. Clarendon Press, Oxford, 1985.

[WHC85] R. West, P. Howell, and I. Cross. *Modelling perceived musical structures. In Howell, Cross, and West (Eds.), Musical structure and cognition*. Academic Press, Londres, 1985.

[Wik13] Wikipedia. The Levenshtein distance. http://en.wikipedia.org/wiki/Levenshtein_distance, consultado en octubre de 2013.