UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

"MEDIDA DE SIMILITUD PARA FORMAS DISCRETAS USANDO UN CÓDIGO CADENA DE PENDIENTE ACUMULADA"

T E S I S

QUE PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS
(COMPUTACIÓN)

P R E S E N T A:

EDUARDO RAMÓN LEMUS VELÁZQUEZ

DIRECTOR DE TESIS: DR. ERNESTO BRIBIESCA CORREA

MÉXICO, D.F. 2008.

A mi familia
y amigos

Declaración

El trabajo intelectual presentado en esta tesis es el resultado de la investigación y esfuerzo personal del autor en sus estudios de posgrado. Ninguna parte de esta tesis ha sido enviada previamente a otra entidad bajo ninguna circunstancia, con excepción de aquellas partes cuya referencia es citada.

Agradecimientos

Agradezco a todas las personas que contribuyeron en la realización de este trabajo. A mi familia por el apoyo brindado, a mis padres Eduardo y Laura, a mis hermanos Antonio y Mónica, y a mi persona especial Ana Patricia Gómez.

Agradezco también las ideas compartidas de amigos y compañeros, en especial de Michel Abdul-Massih, Manuel Arriola, Katya Berrocal, Rubén Durán, Gabriel Hernández, Mauricio Hernández, Victor López, Elias Rivera y Miguel Ruiz-Velasco.

Mis más sinceros agradecimientos y admiración hacía el Dr. Ernesto Bribiesca por haberme compartirdo de su tiempo y experiencia. También, a todos los profesores que con su cátedra han despertado inspiración en mí, principalmente al Dr. Boris Escalante, Dr. Edgar Garduño y Dr. Jorge Urrutia.

A los sinodales Dr. Fernando Arámbula, Dr. Boris Escalante, Dra. Elena Martínez y Dr. Jesús Savage por el interés y valiosos comentarios vertidos hacía este trabajo, y al Dr. Sadegh Abbasi por el material proporcionado.

Finalmente, termino por agradecer a mis compañeros y colegas de la Universidad Panamericana por todas las consideraciones que han tenido para alentar mi desarrollo personal, y al Consejo Nacional de Ciencia y Tecnología por haber ayudado en el financiamiento de mis estudios.

Índice general

Índice de figuras

Índice de cuadros

Medida de similitud para formas discretas usando un código cadena de pendiente acumulada

Eduardo Ramón Lemus Velázquez

Tesis de Maestría en Ciencias de la Computación
Noviembre, 2008

Resumen

Se propone una medida de similitud entre formas 2D orientadas compuestas de celdas regulares y representadas por un nuevo código de cadena. Este código representa el contorno de formas discretas mediante la pendiente acumulada hasta cada uno de los vértices sobre el contorno y es llamado Código Cadena de Pendiente Acumulada (*Slope Accumulation Chain Code, SACC*). Se presentan algunas de las propiedades del código y se incluye una propuesta de métodos para rotar, escalar y hacer transformaciones en espejo para formas representadas por su código. Posteriormente, se hace un análisis de los principales métodos de alineamiento de cadenas basados en Programación Dinámica y a partir de ellos se genera una medida de similitud. Finalmente, con el fin de ilustrar la medida de similitud se presentan algunos de los resultados usando formas reales. Este trabajo está motivado por la idea de convertir un problema de emparejamiento de formas (*shape matching*) en uno más simple de emparejamiento de cadenas (*string matching*) en el que no se requiera la representación de la imagen en el plano cartesiano, significando un ahorro en tiempo de procesamiento y en espacio de almacenamiento.

Capítulo 1

Introducción

A la fecha es clara la dependencia que existe hacía las bases de datos dadas las grandes ventajas que éstas representan en el almacenamiento de grandes volúmenes de datos, la ejecución de búsquedas considerablemente más rápidas, los eficientes mecanismos con los que pueden representar relaciones y además muestran una interfaz uniforme mediante el uso de lenguajes para el almacenamiento y recuperación.

En un primer plano parece cubierto un buen porcentaje de los requerimientos del manejo de datos, sin embargo, existen aun muchas áreas que quedan por ser explotadas y un caso concreto es el manejo de imágenes en bases de datos.

El área de imágenes al momento ha sido muy estudiada y frecuentemente hay mejoras a los procesos de extracción de información. La representación de la forma de los objetos es uno de estos procesos muy importantes que siempre han sido tema de estudio en Visión Computacional.

Entre las representaciones de formas 2D se encuentra uno de los códigos de cadena más citados en la actualidad propuesto por Bribiesca [1,2] y publicado en 1999 llamado Código Cadena Vértice (*Vertex Chain Code, VCC*), el cual permite analizar mucha de la información de la forma de un objeto sin tener que ir a su representación en el plano cartesiano. Entre las propiedades que destacan de este código está el hecho de ser invariante a la rotación, a la traslación y de ser necesario al punto de inicio de la cadena. Muchas aplicaciones interesantes usando representaciones de

cadena en el reconocimiento de objetos han sido reportadas en distintos trabajos.

Freeman [3] plantea que en general, un esquema de codificación para estructuras de línea debe satisfacer tres objetivos:

1. La fiel preservación de la información de interés;

2. El almacenamiento compacto y conveniente que después permita el ser desplegado; y

3. La facilidad para cualquier procesamiento que se requiera.

Existen diversos códigos de cadena que satisfacen estas tres condiciones y *VCC* es uno de ellos. El objetivo planteado en este trabajo consiste en usar estas representaciones lineales para decidir que tan parecidas son las formas discretas que estos códigos representan.

En realidad existen ya muchas técnicas que dan aproximaciones a un resultado de similitud entre formas discretas bajo distintas circunstancias [4,5]. Esas técnicas van desde algo muy simple como encontrar la máxima correlación entre dos matrices que representan la imagen, hasta cosas más complejas como la aproximación de B-splines a segmentos del contorno, pero en cualquier caso trabajando directamente sobre la representación de la imagen en el plano cartesiano. Lo que se intenta lograr en este trabajo es hacer dicha aproximación sin requerir del almacenamiento y manipulación directo de la imagen debido a la complejidad que agrega al procesamiento y al gran espacio que requiere el almacenamiento de muchas imágenes. Para ello, se parte del hecho de que una cadena puede tener concentrada una gran cantidad de información del objeto que se está analizando. En una cadena estará concentrada la información del contorno de una forma discreta y sobre ella se estarán encontrando características representativas del objeto.

En el Capítulo 2 se presenta un nuevo código de cadena con algunas de sus principales características y que presenta algunas ligeras ventajas sobre *VCC* para la

comparación de formas discretas. Además se describen algunos algoritmos de transformación que permiten rotar, trasladar y obtener el reflejo de una forma discreta representada por dicho código.

Posteriormente, se presentan las bases de los algoritmos de alineamiento entre cadenas. El problema planteado de encontrar un alineamiento óptimo entre cadenas, comúnmente usado en áreas de la Biología para comparar cadenas de ADN, es equivalente al problema de encontrar la máxima similitud entre cadenas. Estos algoritmos de similitud se describen con más detalle en el Capítulo 3.

Finalmente, en el Capítulo 4 se aplican los algoritmos anteriormente descritos para definir la similitud entre códigos cadena que representan formas discretas, esperando que se cumpla que a mayor similitud de la cadena, mayor sea la similitud entre los objetos descritos por esos códigos.

La solución propuesta queda específicamente limitada al trabajo con el contorno del objeto mediante su código de cadena.

Puntualizando la aportación de este trabajo, se tiene entonces que las dos principales contribuciones son, por un lado la presentación de un nuevo código de cadena para representar formas discretas describiendo algunas de sus propiedades, y por el otro lado la extracción de una medida de similitud entre códigos cadena usando algoritmos de alineamiento óptimo.

Capítulo 2

Código Cadena de Pendiente Acumulada (SACC)

En esta sección se presenta el Código Cadena de Pendiente Acumulada (*Slope Accumulation Chain Code, SACC*), como obtenerlo y algunas de sus principales características. Una importante simplificación está en asumir que se trabaja con objetos que ya han sido aislados del resto de la imagen mediante un proceso eficiente de segmentación y que posteriormente fueron binarizados. Estas entidades, llamadas *formas discretas* [1], son formadas por celdas regulares. La Figura 2.1 muestra una forma discreta compuesta de celdas de forma (a) rectangular, (b) hexagonal y, (c) triangular. En el contenido de este trabajo, la longitud l de cada lado de la celda regular será igual a 1.

La frontera de una forma discreta compuesta de celdas regulares puede ser representada por su código de cadena. Considerando que este documento se concentra en trabajar con formas, se manejarán curvas cerradas aunque ésto no limita a que se pueda representar cualquier tipo de curva discreta.

Para el caso de celdas rectangulares, otra simplificación asumida es la de estar trabajando con figuras 4-conectadas sin hoyos, excluyendo así algunos problemas comúnmente presentados en la literatura [6].

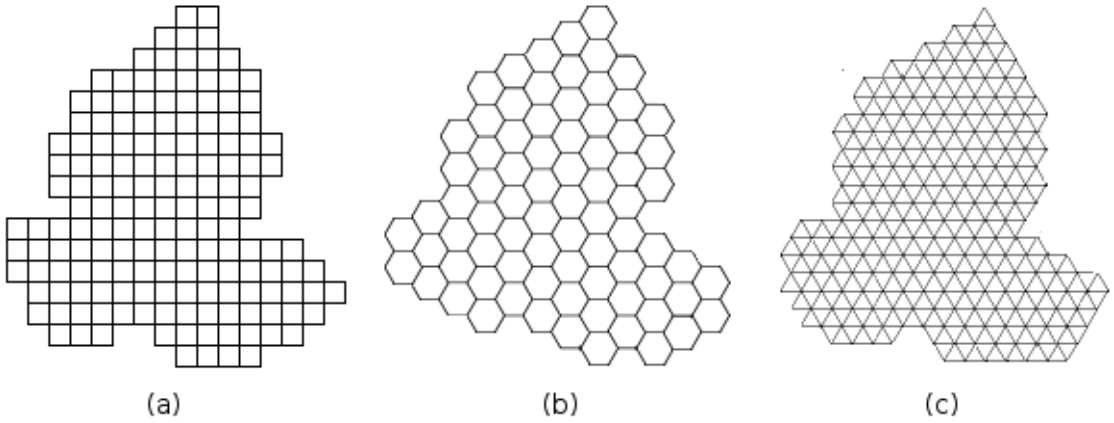


Figura 2.1: Formas discretas compuestas de celdas regulares: (a) rectangulares; (b) hexagonales; (c) triangulares

2.1. Definiciones

2.1.1. Elemento de una cadena

Un elemento a_i de una cadena indica el valor de una pendiente acumulada hasta el i-ésimo vértice ubicado sobre el contorno de la forma discreta. Esto nos sugiere entonces, que existe una dependencia al punto de inicio en donde el valor acumulado de la pendiente empieza siendo 0. Más adelante se mencionan a detalle aspectos relacionados al punto de inicio para figuras orientadas.

El valor de incremento en la pendiente de la i-ésima posición es obtenido con la fórmula:

$$slope_i = \frac{T}{2} - B_i \tag{2.1}$$

donde $slope_i$ es el incremento a la pendiente en el i-ésimo vértice sobre el contorno, B_i es la cantidad de celdas del objeto en contacto con el i-ésimo vértice y T es la cantidad de celdas totales en contacto con cualquier vértice de la malla (Figura 2.2).

Figura 2.2: Los valores de incremento para los vértices marcados son: (a) -1; (b) -0.5; (c) -1. Las celdas del objeto se muestran sombreadas y las del fondo en blanco

En general, el valor de cualquier elemento a_i de la cadena $SACC$ se puede encontrar con la fórmula:

$$a_i = \begin{cases} slope_i & i = 1 \\ slope_i + a_{i-1} & i > 1 \end{cases} \tag{2.2}$$

Las Figuras 2.3(a-c) muestran 3 formas discretas sobre los 3 tipos de celdas regulares. Para cada una de estas formas, se marca sobre cada vértice del contorno el valor de la cadena $SACC$ que correspondería de acuerdo a cada tipo de celda.

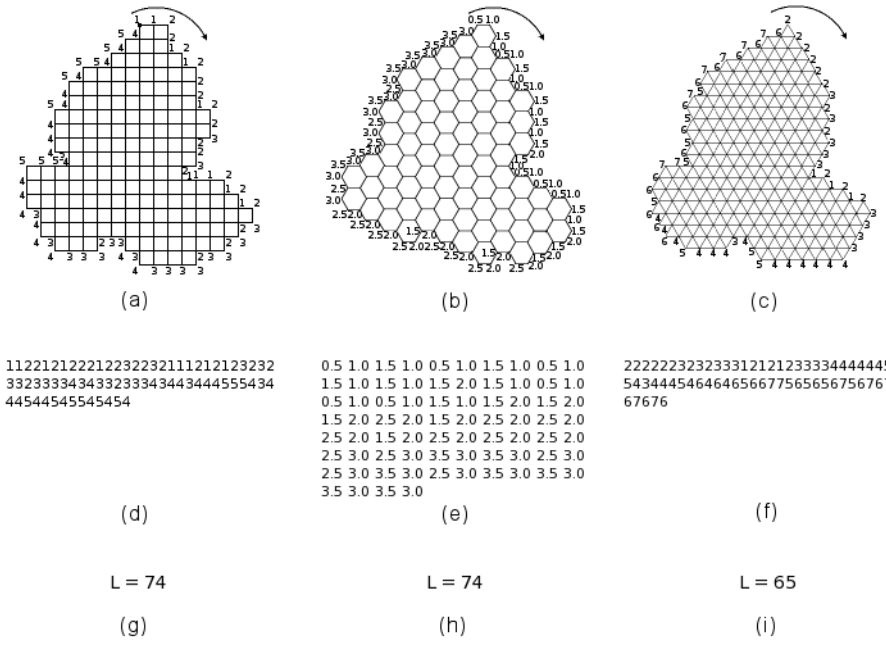


Figura 2.3: (a-c) Formas discretas con su código $SACC$; (d-f) Código $SACC$ recorrido en sentido de las manecillas del reloj; (g-i) Longitudes de los códigos

2.1.2. Una cadena

Una cadena A es una secuencia ordenada de elementos y queda definida por:

$$A = a_1 a_2 \dots a_n = \{a_i \ : \ 1 \leq i \leq n\} \tag{2.3}$$

donde n es la cantidad de elementos de la cadena. Al ser una secuencia ordenada, será importante la selección del sentido de recorrido de la forma discreta para la obtención de la cadena. De las formas discretas de las Figuras 2.3(a-c) se generan las cadenas $SACC$ de las Figuras 2.3(d-f) usando el orden de recorrido en sentido de la manecillas del reloj y que es el mismo orden que se usará en el resto del documento.

2.1.3. La longitud de una cadena

La longitud L de una cadena es la suma de las longitudes de sus elementos:

$$L = nl \tag{2.4}$$

donde n es la cantidad de elementos de la cadena y l la longitud de una arista de la celda regular.

Para el caso de las Figuras 2.3(a-c), sus longitudes aparecen en las Figuras 2.3(g-i) respectivamente, asumiendo como ya se había dicho, que las celdas regulares tienen una longitud de 1 en cada uno de sus lados.

2.1.4. Derivada de una cadena

La primera derivada de una cadena $SACC$ A se denotada como $A^{'}$ y tiene en cada uno de sus elementos $a_i^{'}$ la diferencia entre el elemento a_i y el elemento a_{i-1} de A. La longitud de la cadena $A^{'}$ es igual a la longitud de la cadena A.

$$A^{'} = a_1^{'} a_2^{'} \dots a_n^{'} = a_1 \ \ a_2 - a_1 \ \ a_3 - a_2 \ \dots \ a_n - a_{n-1} \tag{2.5}$$

La siguiente fórmula define el valor de cada elemento en A':

$$a_i^{'} = \begin{cases} T + a_1 - a_n & i = 1 \\ a_i - a_{i-1} & 1 < i \leq n \end{cases} \tag{2.6}$$

Es de notar que el primer elemento de la cadena derivada es $a_1 - 0$ puesto que la pendiente acumulada de inicio es 0. Esto ocurre porque para curvas cerradas $T = a_n$ (2.1.5). Es por esto que se pone a a_1 como primer elemento.

La primera derivada de una cadena *SACC* (*SACC'*) representa un código cadena en sí mismo con propiedades muy parecidas a las de *VCC* [1] entre las que destaca la invarianza a la rotación. Además, en la primera derivada de una cadena *SACC*, el valor asignado a cada elemento de la cadena es independiente al punto de inicio desde el que se está calculando su código *SACC*.

La Figura 2.4(a-c) muestra una forma representada sobre celdas regulares rectangulares, hexagonales y triangulares respectivamente y la Figura 2.4(d-f) tiene la primera derivada del código *SACC* de esas cadenas. Dada la similitud en algunas de las propiedades de *VCC* con el código cadena de primera derivada de *SACC* (*SACC'*), es posible establecer una relación que convierta de un código a otro, Figura 2.4(g-i). Primero, para convertir de código *SACC'* a *VCC* se usa la fórmula:

$$vcc_i = \frac{T}{2} - sacc_i^{'} \tag{2.7}$$

donde vcc_i es el i-ésimo elemento de la cadena *VCC* y $sacc_i^{'}$ el i-ésimo elemento de la cadena *SACC'*. En sentido inverso, la fórmula mantiene la misma estructura

$$sacc_i^{'} = \frac{T}{2} - vcc_i \tag{2.8}$$

Finalmente, para poder regresar de una cadena *SACC'* a una cadena *SACC*

$$sacc_i = \sum_{j=1}^{i} sacc_j^{'} \tag{2.9}$$

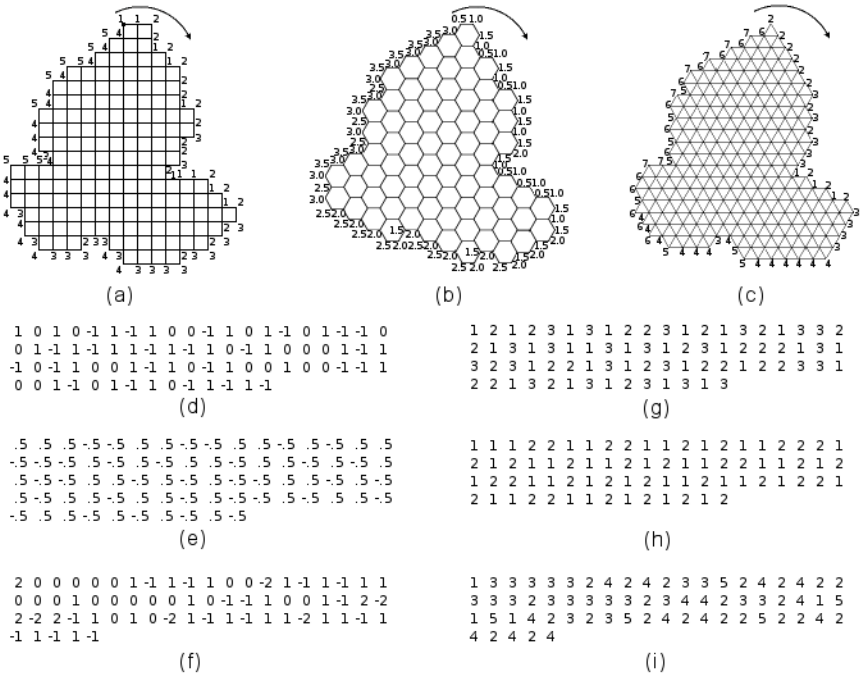


Figura 2.4: (a-c) Formas discretas con su código *SACC*; (d-f) Primeras derivadas *SACC* de los códigos anteriores; (g-i) Código *VCC* de los códigos anteriores

2.1.5. Teorema 1: Último elemento en una curva cerrada

Para cualquier forma discreta compuesta de celdas regulares, la cadena *SACC* que representa su contorno es siempre una curva cerrada y el último elemento de esa cadena es igual a T.

2.1.6. Teorema 2: Ángulo de cambio

Para cualquier forma discreta compuesta de celdas regulares, el ángulo de cambio θ_i en su i-ésimo vértice en el sentido en el que ser recorre el contorno queda definido por la siguiente ecuación y se ilustra en la Figura 2.5:

$$\theta_i = \frac{360^\circ \; slope_i}{T} \tag{2.10}$$

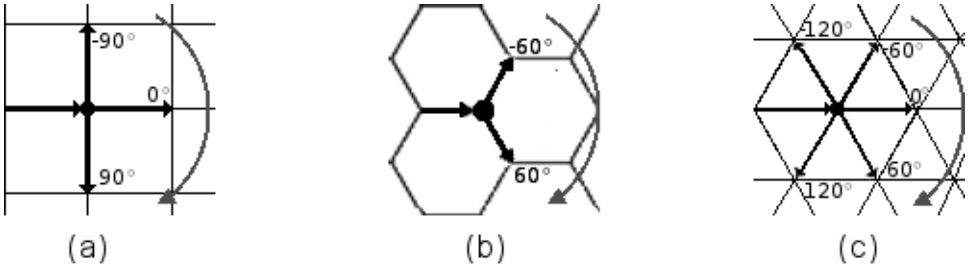


Figura 2.5: Valores del ángulo de cambio en celdas: (a) rectangulares; (b) hexagonales; (c) triangulares

2.2. Nivel de anidamiento

Al igual que el código de cadena propuesto por Freeman (FCC) [3], $SACC$ trabaja sobre figuras orientadas y es sensible al punto de inicio e indirectamente a la rotación. Por esto, el valor de un elemento de la cadena representa una dirección, que a diferencia de FCC donde los valores son arbitrarios, en $SACC$ representan la pendiente acumulada desde un punto determinado hasta ese elemento. Esto permite entre otras cosas, diferenciar entre dos elementos que tienen la misma dirección pero distinto nivel de anidamiento del elemento en el contorno. En la Figura 2.6 se resaltan dos elementos con la misma dirección y que sin embargo tienen valores distintos debido a sus niveles de anidamiento diferentes. El elemento con valor 1 tiene un nivel de anidamiento 1 mientras que el de valor 9 pertenece a un segmento del contorno que entra en una especie de espiral que da dos vueltas completas y se encuentra en la tercera de estas vueltas por lo que tiene un nivel de anidamiento 3.

El nivel de anidamiento se puede obtener por la ecuación:

$$N(a_i) = \begin{cases} \left\lfloor \frac{a_i}{o} \right\rfloor & a < 0 \\ 0 & a_i = 0 \\ \left\lceil \frac{a_i}{o} \right\rceil & a_i > 0 \end{cases} \tag{2.11}$$

donde N es el nivel de anidamiento, a_i el i-ésimo elemento de la cadena y o el número

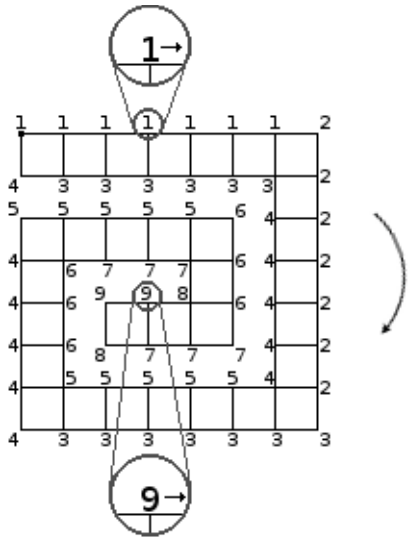


Figura 2.6: Dos elementos de una cadena *SACC* con la misma orientación y distinto valor en *SACC* debido al distinto nivel de anidamiento.

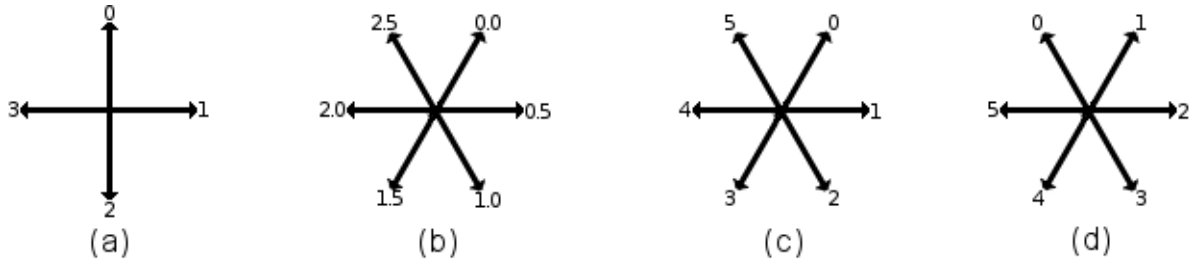


Figura 2.7: Valores de *SACC* en el nivel de anidamiento 1 en celdas: (a) rectangulares; (b) hexagonales; (c) triangulares; (d) triangulares alterno

de orientaciones que puede tener cualquier segmento del contorno sobre una arista de la celda regular: 4 para celdas rectangulares, 6 para celdas hexagonales y 6 para celdas triangulares. Figura 2.7.

Para aumentar el nivel de anidamiento se debe de entrar en una espiral en la que cada 360° se aumenta en 1 el nivel de anidamiento hasta llegar al final de la cadena donde para figuras cerradas vuelve a ser 1 dicho nivel .

Para celdas rectangulares donde una vuelta completa (360°) se logra con 4 incrementos de 1 (90°), basta sumar o restar 4 para aumentar o disminuir en uno

respectivamente el nivel de anidamiento sin modificar la dirección de un elemento.

Para el caso de celdas hexagonales son 6 incrementos de 0.5 (60°), por lo que sumando o restando 3 se modifica en uno el nivel de anidamiento sin modificar la dirección. Finalmente, para celdas triangulares son 3 incrementos de 2 (120°) lo cual índica que sumando o restando 6 se modifica en uno el nivel de anidamiento sin modificar la dirección del elemento.

Para comparar direcciones de elementos en una cadena $SACC$ que probablemente tienen distintos niveles de anidamiento, se debe llevar el valor de elementos al nivel de anidamiento 1 (Figura 2.7) obteniendo de este modo un código similar a FCC.

Es importante notar que para celdas regulares triangulares, al no tratarse de una malla en la que todas las celdas tengan la misma orientación, el vértice de inicio es un píxel con dos posibles orientaciones (Figura 2.8). Para el caso de la Figura 2.8(a), los ejes son como indica la Figura 2.7(c), y para el caso de la Figura 2.8(b), los ejes se recorren 1 incremento como se muestra en la Figura 2.7(d).

Una fórmula para llevar cualquier elemento $SACC$ a un nivel de anidamiento 1 sin modificar su dirección es:

$$a_i = Mod(Mod(a_i, o) + o, o) \tag{2.12}$$

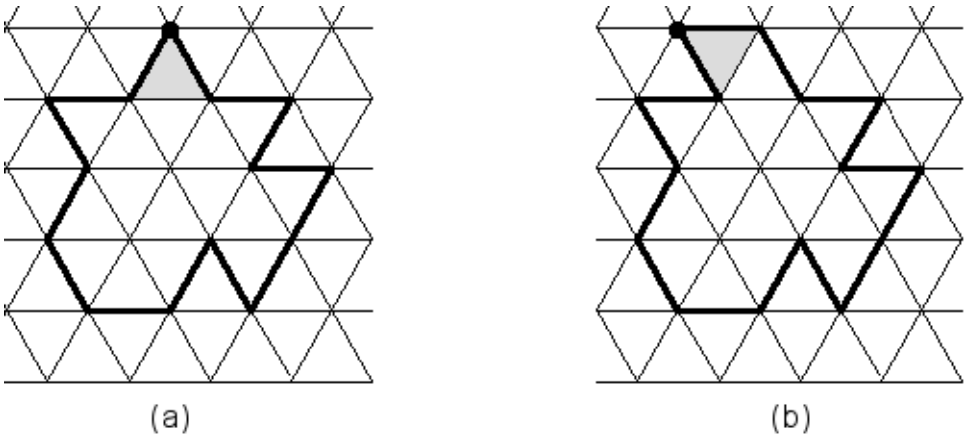


Figura 2.8: Orientación de píxeles en celdas triangulares: (a) orientación 1; (b) orientación 2

2.3. Posición relativa y absoluta en la imagen

A partir de una cadena *SACC* es posible obtener la posición relativa al punto de inicio de cualquiera de los elementos de la cadena. Es decir, saber cuánto habría que desplazarse en el plano cartesiano en cada uno de los ejes de la malla para que, a partir del punto de inicio se llegue a cierto vértice sobre el contorno.

Para esto, se establece un sentido en los ejes. La Figura 2.9 muestra la notación que se ha escogido para los ejes en este documento. Por ejemplo, en la Figura 2.9(a) para el *Eje 1-3*, cuando el nivel de anidamiento 1 de un elemento a_i de la cadena situado sobre un vértice v con coordenadas (v_{Eje1-3}, v_{Eje0-2}) sea 1, habrá un desplazamiento positivo de una unidad en ese eje y sabremos entonces que el vértice del elemento a_{i+1} está en la posición $(v_{Eje1-3} + 1, v_{Eje0-2})$. Si por el contrario, el nivel de anidamiento 1 del elemento a_i de la cadena es 3, habrá un desplazamiento negativo de una unidad sobre el mismo *Eje 1-3 y* el vértice del elemento a_{i+1} estaría en la posición $(v_{Eje1-3} - 1, v_{Eje0-2})$. Algo equivalente ocurre para el *Eje 0-2*.

Debido a que para celdas triangulares y hexagonales los vectores base no son ortonormales, cuando se desplace sobre un eje, existirá también un desplazamiento sobre los demás ejes producto de la proyección del desplazamiento original sobre el resto de los ejes. El Cuadro 2.1 muestra los desplazamientos que se deben hacer sobre cada eje en función del elemento de la cadena en su nivel de anidamiento 1.

En la Figura 2.10 se muestra una forma discreta con su código *SACC* y la posicion relativa a un punto de inicio de cada uno de sus vértices.

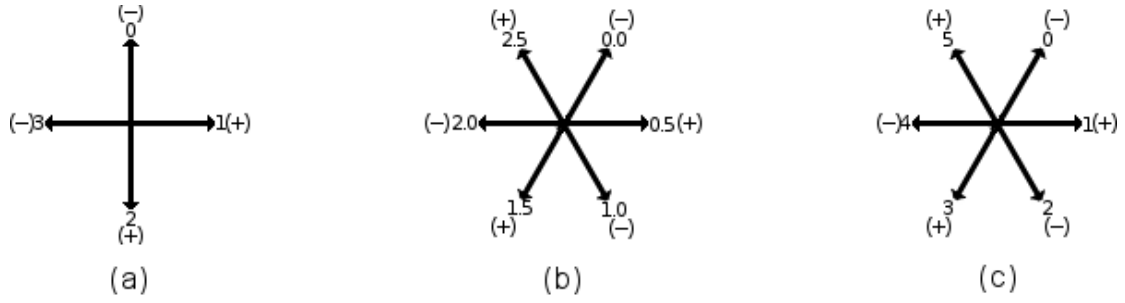


Figura 2.9: Convención en el sentido de los ejes de direcciones *SACC* para celdas: (a) rectangulares; (b) hexagonales; y (c) triangulares

	Eje 1-3	*Eje 0-2*
0	0	1
1	1	0
2	0	-1
3	-1	0

(a)

	Eje 0.5-2	*Eje 1-2.5*	*Eje 0-1.5*
0	0.5	0.5	-1.0
1	1.0	-0.5	-0.5
2	0.5	-1.0	0.5
3	-0.5	-0.5	1.0
4	-1.0	0.5	0.5
5	-0.5	1.0	-0.5

(b)

	Eje 1-4	*Eje 2-5*	*Eje 0-3*
0	0.5	0.5	-1.0
1	1.0	-0.5	-0.5
2	0.5	-1.0	0.5
3	-0.5	-0.5	1.0
4	-1.0	0.5	0.5
5	-0.5	1.0	-0.5

(c)

Cuadro 2.1: Incrementos en la posición para desplazamientos sobre celdas: (a) rectangulares; (b) hexagonales; y (c) triangulares

La posición relativa de una forma discreta orientada cambia de acuerdo al punto de inicio que se seleccione. Una forma simple de lograr la invarianza de la posición de un elemento en la cadena es mediante una posición absoluta. La posición absoluta toma un marco de referencia en el que el valor mínimo de una forma discreta en todos los ejes coincide con el valor 0 de cada uno de esos ejes:

$$pos^*_{EjeX}(a_i) = pos_{EjeX}(a_i) - min(pos_{EjeX}(a_1), \ldots, pos_{EjeX}(a_n)) \tag{2.13}$$

donde $pos^*_{EjeX}(a_i)$ es la posición absoluta en el *Eje X* para el elemento i de la cadena A y $pos_{EjeX}(a_x)$ es la posición relativa al inicio en el *Eje X* para el elemento x de la cadena A.

Por ejemplo, en la Figura 2.10 el valor de posición relativa mínimo en el *Eje 1-3* entre todos los elementos es -2. Entonces, para obtener las posiciones absolutas en ese eje a todos los elementos hay que restarles -2. La Figura 2.11 muestra las posiciones absolutas para las formas discretas de la Figura 2.10.

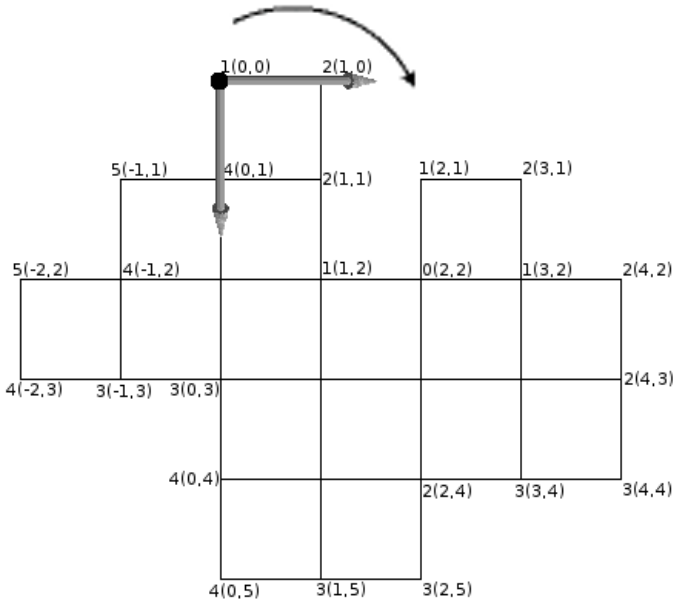


Figura 2.10: Forma discreta con las posiciones relativas de cada uno de sus vértices

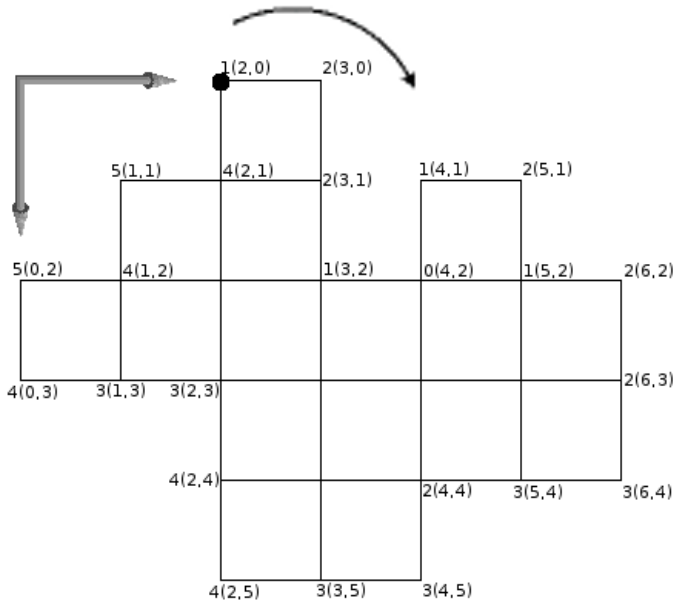


Figura 2.11: Forma discreta con las posiciones absolutas de cada uno de sus vértices

Esto también permite saber algunas otras propiedades de la imagen como su ancho W:

$$W = max(pos^*_{Eje1-3}(a_i)) = max(pos_{Eje1-3}(a_i)) - min(pos_{Eje1-3}(a_i)) \tag{2.14}$$

y el alto H:

$$H = max(pos^*_{Eje0-2}(a_i)) = max(pos_{Eje0-2}(a_i)) - min(pos_{Eje0-2}(a_i)) \tag{2.15}$$

Para el caso de la Figura 2.11 $W = 6$ y $H = 5$.

A partir de este momento, con el fin de acotar y simplificar las descripciones, se tratarán exlusivamente características de formas discretas representadas sobre celdas regulares con forma rectangular.

2.4. Independencia a la traslación

SACC es invariante a la traslación debido a que representa el valor de una pendiente acumulada a lo largo de cada uno de los vértices del contorno de la forma discreta sin importar donde se localiza ésta. Entonces cada elemento de una cadena es obtenido sin considerar en ningún momento la posición del objeto con respecto a la imagen.

2.5. Punto de inicio

Usando *SACC* para representar el contorno de formas discretas orientadas compuestas de celdas regulares, todas las cadenas son cerradas. El punto de inicio de una cadena puede ser normalizado considerando una jerarquía en los ejes y una posición dentro de ellos.

	Jerarquía Primaria	Jerarquía Secundaria
1	$min(pos_{Eje0-2}(a_i))$	$min(pos_{Eje1-3}(a_i))$
2	$min(pos_{Eje0-2}(a_i))$	$max(pos_{Eje1-3}(a_i))$
3	$max(pos_{Eje1-3}(a_i))$	$min(pos_{Eje0-2}(a_i))$
4	$max(pos_{Eje1-3}(a_i))$	$max(pos_{Eje0-2}(a_i))$
5	$max(pos_{Eje0-2}(a_i))$	$max(pos_{Eje1-3}(a_i))$
6	$max(pos_{Eje0-2}(a_i))$	$min(pos_{Eje1-3}(a_i))$
7	$min(pos_{Eje1-3}(a_i))$	$max(pos_{Eje0-2}(a_i))$
8	$min(pos_{Eje1-3}(a_i))$	$min(pos_{Eje0-2}(a_i))$

Cuadro 2.2: Definición de ocho puntos notables en una forma discreta

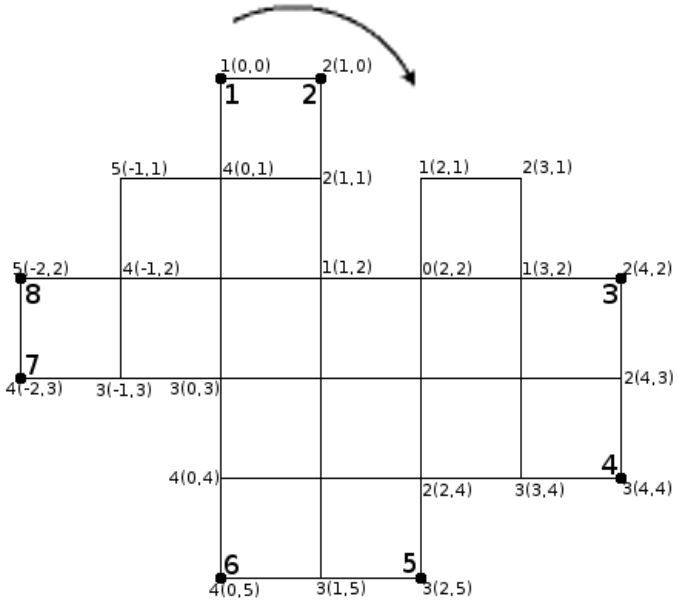


Figura 2.12: Ocho puntos notables en el contorno de una forma discreta

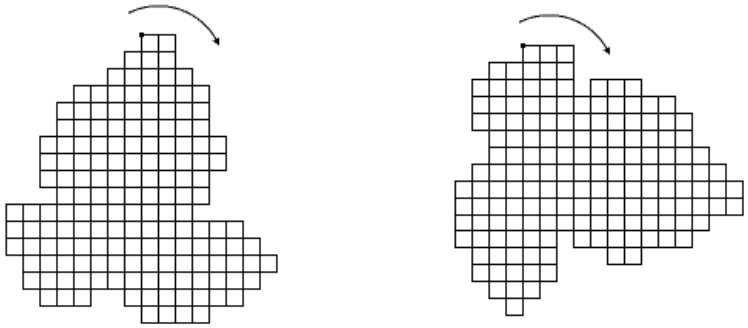


Figura 2.13: Figuras con su punto de inicio normalizadas

Para una forma discreta representada sobre celdas regulares rectangulares, podríamos definir como notables los ocho puntos descritos en el Cuadro 2.2. Esos puntos obedecen a una posición dentro de cada uno de los ejes y la cumplen asignando una jerarquía a cada uno de esos ejes. Cada uno de esos ocho puntos siempre existe dentro de la cadena y se destacan en la Figura 2.12.

Para el caso del punto notable 1, por ejemplo, podemos ver que corresponde al vértice sobre el contorno del objeto con la posición mínima en el *Eje 0-2* y en caso de haber más de un elemento con el mismo valor en dicho eje se elige al que tiene la posición mínima en el *Eje 1-3*.

Teniendo una serie de puntos que siempre pueden ser identificados en el contorno de una forma discreta, resulta posible elegir uno de ellos como punto de inicio logrando normalizar el punto de inicio en la descripción de cualquier forma discreta orientada.

Particularmente, en este trabajo el punto notable 1 será reconocido como el inicio de una cadena. Dicho de forma distinta, se está escogiendo como punto de inicio el primer vértice sobre el contorno del objeto que se encuentra barriendo la imagen de arriba a abajo y de izquierda a derecha. La Figura 2.13 nos muestra algunas formas discretas orientadas ya normalizadas en su punto de inicio.

2.6. Rotación

Al estar describiendo formas discretas orientadas se entiende que habiendo definido un punto de inicio, $SACC$ no es invariante a la rotación, sino que por el contrario, en la orientación de la forma discreta encuentra más información del objeto.

En ocasiones será útil rotar el objeto e idealmente se quiere hacer directo sobre su cadena. $SACC$ soporta rotaciones en incrementos iguales a $\frac{360^\circ}{o}$ donde o ya había definido como el número de orientaciones que puede tener un elemento de la cadena.

Para el caso de celdas regulares rectangulares sabemos que $o = 4$ ya que existen 4 posibles orientaciones para un elemento de la cadena $SACC$: arriba, abajo, izquier-

da y derecha. Esto significa que sobre celdas regulares rectangulares se soportan rotaciones en incrementos de 90°.

Supongamos que se tiene una cadena A de n elementos que representa una forma discreta sobre celdas regulares rectangulares, y que se quiere rotar un ángulo α (múltiplo del incremento) en sentido contrario al que se obtuvo su cadena $SACC$.

Previo al proceso de rotación hay que seguir dos pasos de preprocesamiento que básicamente consisten en asegurarse de tener un valor adecuado de α (punto 1) y tener un arreglo R con la posición de los potenciales nuevos puntos de inicio (punto 2). Los pasos a seguir para lograr la rotación son tres (puntos 3-5) y se detallan a continuación:

1. Se debe normalizar el número de incrementos a rotar llevándolo a un nivel de anidamiento 1. Esto da como resultado un número de incrementos k tal que $0 \leq k < o$. Es importante notar que si $k = 0$, la rotación ha sido terminada porque se intenta rotar un múltiplo de 360° lo cual no afecta la orientación del objeto. La fórmula para normalizar el ángulo α de rotación es

$$k = Mod\left(Mod\left(\frac{\alpha o}{360^\circ}, o\right) + o, o\right) \tag{2.16}$$

2. Obtener un arreglo unidimensional R de longitud o con las posiciones de los elementos previos a los posibles puntos de inicio. Para el caso de celdas rectangulares los 4 valores del arreglo están definidos en el Cuadro 2.3 y corresponden a los elementos previos a los puntos notables 1, 3, 5 y 7 considerando que la cadena $SACC$ fue obtenida en sentido de las manecillas del reloj. La Figura 2.14 muestra resaltados con una flecha esos 4 elementos en el objeto.

3. Restar k a todos los elementos de la cadena.

4. Sumar o únicamente a los primeros $R[k]$ elementos de la cadena.

5. Finalmente, en la cadena hay que hacer un desplazamiento circular a la izquierda de $R[k]$ posiciones.

Índice	Valor
0	Índice del elemento previo al punto notable 1
1	Índice del elemento previo al punto notable 3
2	Índice del elemento previo al punto notable 5
3	Índice del elemento previo al punto notable 7

Cuadro 2.3: Arreglo R con los elementos previos a los posibles puntos de inicio

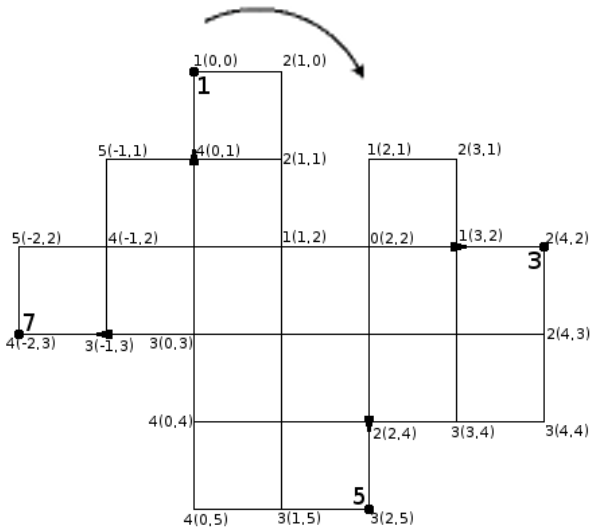


Figura 2.14: Elementos del Cuadro 2.3 marcados con una flecha

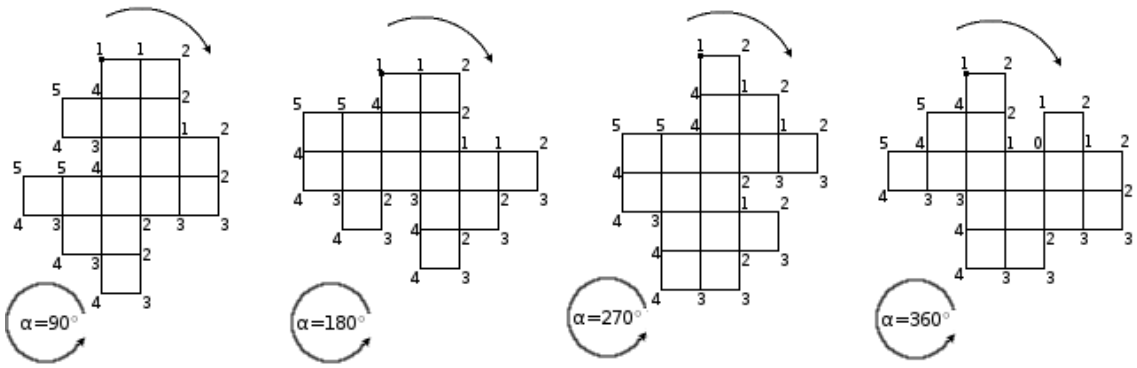
El valor de cualquier elemento de la cadena después de haber sido rotado queda definido con la siguiente fórmula:

$$a_i^{'} = \begin{cases} a_i, & k = 0 \\ a_{Mod(i+R[k]-1,n)+1} - k, & i \leq n - R[k] \\ a_{Mod(i+R[k]-1,n)+1} - k + o, & i > n - R[k] \end{cases} \tag{2.17}$$

El Algoritmo 1 describe la forma de rotar una forma representada por su código *SACC,* y finalmente, la Figura 2.15 muestra las cuatro posibles rotaciones que se pueden hacer a la Figura 2.12 así como los pasos para lograrlo.

Algoritmo 1 Rotación de una forma discreta representada por su código *SACC*

```
ROTATE(A, alpha)
   n ← Length(A)
   k ← Mod(Mod(alpha * 4 / 360, 4) + 4, 4)
   FillRArray(R)
   for i ← 1 to n do
      A[i] ← A[i] − k
      if i <= R[k] then
         A[i] ← A[i] + 4
   lshift(A, R[k])
```



1. k = 1
2. R[k] = 8
3. A = 0 1 1 0-1 0 1 0 1 1
2 2 1 2 2 3 3 2 2 3
4 3 4 3
4. A = *4 5 5 4 3 4 5 4* 1 1
2 2 1 2 2 3 3 2 2 3
4 3 4 3
5. A = 1 1 2 2 1 2 2 3 3 2
2 3 4 3 4 3 4 5 5 4
3 4 5 4

1. k = 2
2. R[k] = 13
3. A = -1 0 0-1-2-1 0-1 0 0
1 1 0 1 1 2 2 1 1 2
3 2 3 2
4. A = *3 4 4 3 2 3 4 3 4 4*
5 5 4 1 1 2 2 1 1 2
3 2 3 2
5. A = 1 1 2 2 1 1 2 3 2 3
2 3 4 4 3 2 3 4 3 4
4 5 5 4

1. k = 3
2. R[k] = 19
3. A = -2-1-1-2-3-2-1-2-1-1
0 0-1 0 0 1 1 0 0 1
2 1 2 1
4. A = *2 3 3 2 1 2 3 2 3 3*
4 4 3 4 4 5 5 4 4 1
2 1 2 1
5. A = 1 2 1 2 1 2 3 3 2 1
2 3 2 3 3 4 4 4 4 4
5 5 4 4

1. k = 0

Figura 2.15: Rotaciones de la forma discreta de la Figura 2.14 sobre celdas rectangulares

2.7. Transformaciones en espejo

Usando *SACC* para formas orientadas compuestas de celdas regulares rectangulares es posible hallar el simétrico en el *Eje 1-3* (simétrico horizontal) directamente sobre la cadena de una manera relativamente simple.

Para ello hay que conservar en la cadena A el prefijo de q valores 1, siendo a_{q+1} el primer elemento distinto de 1 en la cadena A. El resto de los elementos se le restan a 6 y son ordenados de forma inversa a como aparecen en la subcadena $a_{q+1}a_{q+2}\ldots a_n$. Es decir, el resultado será un prefijo de q valores iguales a 1 seguido de una subcadena $(6-a_n)(6-_{n-1})\ldots(6-a_{q+1})$.

Entonces la fórmula que define como obtener el simétrico sobre el *Eje 1-3* para formas discretas sobre celdas regulares rectangulares es

$$a_i^M = \begin{cases} 1, & i \leq q \\ 6 - a_{n+q+1-i}, & i > q \end{cases} \tag{2.18}$$

El Algoritmo 2 describe la obtención del espejo horizontal de una forma descrita con *SACC*. La Figura 2.16 muestra el simétrico sobre el *Eje 1-3* de una forma discreta. Obtener el simétrico sobre el *Eje 1-3* y después sobre el *Eje 0-2* es equivalente a rotar 180° mientras que el simétrico únicamente sobre el *Eje 0-2* es el simétrico sobre el *Eje 1-3* del resultado de la rotación anterior (Figura 2.17).

Algoritmo 2 Reflejo horizontal de una forma discreta representada por su código *SACC*

```
H_MIRROR(A)
   q ← 0
   while A[q + 1] = 1 do
      q ← q + 1
   w ← n + q
   for i ← q + 1 to Floor(w / 2) do
      tmp ← 6 − A[i]
      A[i] ← 6 − A[w + 1 − i]
      A[w + 1 − i] ← tmp
```

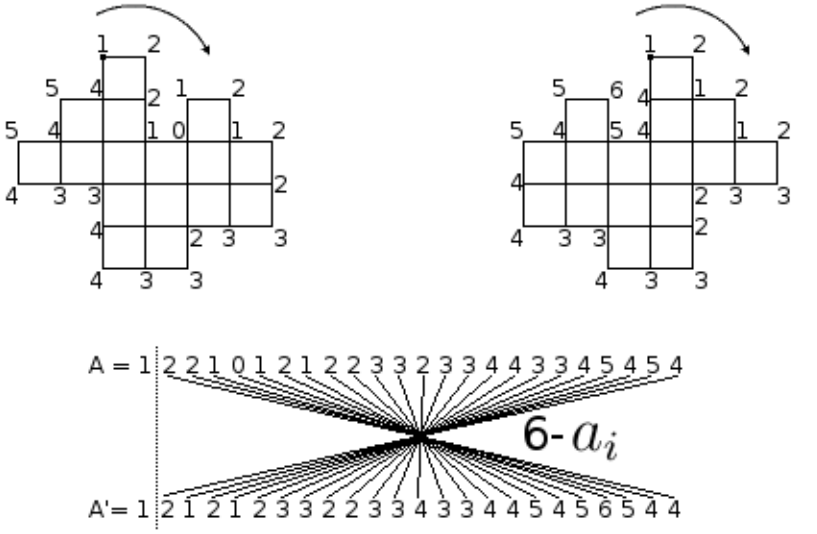


Figura 2.16: Simétrico sobre el *Eje 1-3* para una forma discreta

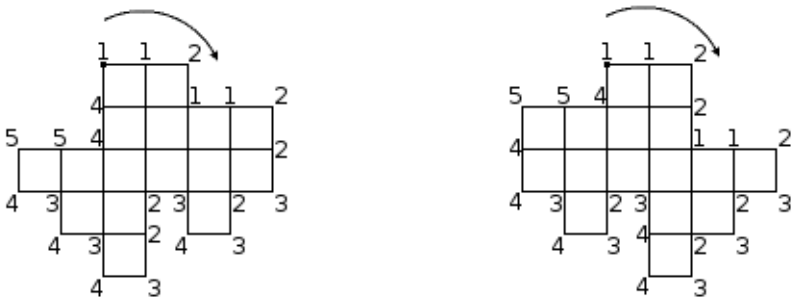


Figura 2.17: Simétricos también sobre el *Eje 0-2* para una forma discreta

2.8. Escalamiento

Una debilidad que tiene en general cualquier código de cadena es su sensibilidad a la escala. En la mayoría de los casos es muy complicado establecer la relación que existe entre dos códigos de cadena que representan la misma forma discreta a distintas escalas.

SACC es un código de cadena que resulta sensible a la escala. Es decir, el código *SACC* de una forma discreta a cierta escala es distinto del código *SACC* de la misma

Algoritmo 3 Algoritmo para escalar un código SACC en un factor r >1

```
SCALE_UP(A, length, S, slength)
   S[0] ← A[0]
   lastVal ← A[0]
   idx ← 1
   lastY ← Round (pos*_Eje0−2 (A[0]) * ry)
   for j ← 1 to length do
      i ← Mod (j, length)
      switch (Mod (Mod (lastVal, 4) + 4, 4))
         case 0    %Va hacía arriba
            lastY ← lastY + 1
            while (lastY < Round (pos*_Eje0−2 (A[0]) * ry) do
               S[idx] ← lastVal
               idx ← idx + 1
               lastY ← lastY + 1
         case 2    %Va hacía abajo
            lastY ← lastY − 1
            while (lastY > Round (pos*_Eje0−2 (A[0]) * ry) do
               S[idx] ← lastVal
               idx ← idx + 1
               lasry ← lastY − 1
      if i > 0
         lastVal ← A[i]
         S[idx] ← A[i]
      idx ← idx + 1
   slength ← idx − 1
```

forma discreta a una escala distinta, y resulta muy díficil establecer algún tipo de relación directa entre estas cadenas.

A continuación se describe un algoritmo que permite escalar una forma discreta representada por su código *SACC* en uno de los ejes coordenados por un factor $r > 1$ (Algoritmo 3).

Este mismo algoritmo puede adaptarse para escalar también el otro eje coordenado o puede volver a ser usado sin ninguna adaptación sobre el resultado del escalamiento rotado 90° lo cual dará como resultado una figura escalada en ambos ejes.

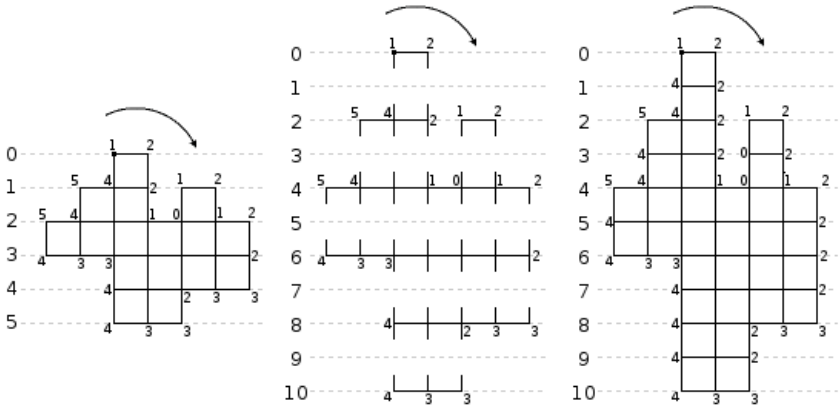


Figura 2.18: Proceso de escalamiento en un factor $r_y = 2$

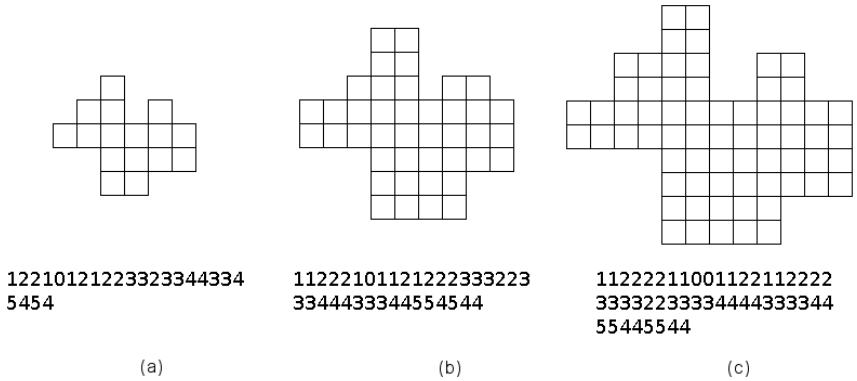


Figura 2.19: Escalamiento en factores r_x y r_y igual a: (a) 1.0; (b) 1.5; (c) 2.0

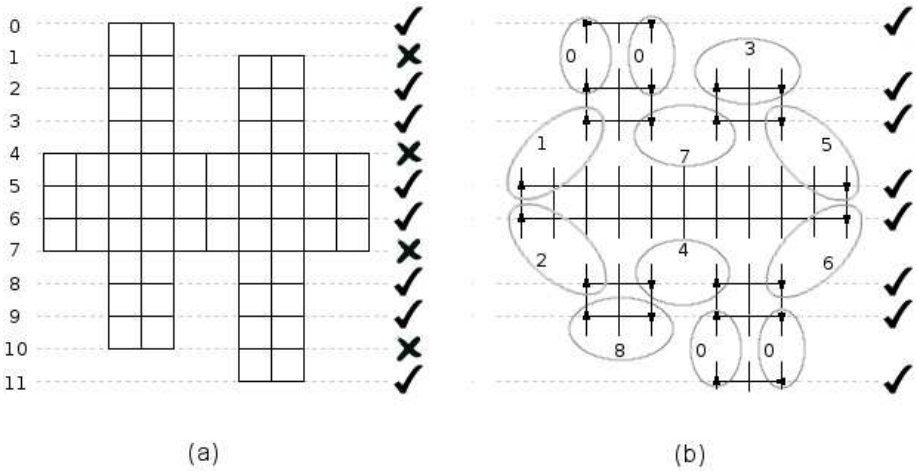


Figura 2.20: Proceso de escalamiento en un factor $r_y = \frac{2}{3}$

El algoritmo se basa en la idea de incrementar proporcionalmente en un eje la posición absoluta de los puntos de la imagen original. Es decir, un elemento a_i de la cadena con vértice en la coordenada ($pos^*_{Eje1-3}(a_i)$, $pos^*_{Eje0-2}(a_i)$) de la forma discreta original, tendrá su posición en ($pos^*_{Eje1-3}(a_i)$, $Round(pos^*_{Eje0-2}(a_i) * r_y)$) de la forma discreta ya escalada por un factor r_y en el *Eje 0-2*.

Una de las ventajas del algoritmo es trabajar directamente sobre la cadena $SACC$ sin recurrir a ninguna operación en el plano cartesiano, lo cual permite lograr el escalamiento con un procesamiento muy ligero $\Theta(n)$ siendo n la cantidad de vértices sobre el contorno.

El espacio entre el elemento a_{i-1} y el elemento a_i ahora será ocupado por $Round(pos^*_{Eje0-2}(a_i) * r_y) - pos^*_{Eje0-2}(a_i)$ elementos a_{i-1} que corresponden a los puntos agregados producto del incremento del tamaño en un eje (Figura 2.18).

La Figura 2.19 muestra formas discretas escaladas por el algoritmo de incremento dando un resultado similar al de una interpolación de orden cero o también conocida como interpolación del vecino más cercano.

Para el caso de un factor de escalamiento $r < 1$, a continuación se propone un algoritmo que se centra en la misma idea de manipular directamente las posiciones absolutas en la cadena, pero con la diferencia que ahora se eliminan elementos a_i de la cadena cuyas posiciones absolutas en el *Eje 0-2* sean distintas a $Round(Round(pos^*_{Eje0-2}(a_i) * r_y)/r_y)$. Figura 2.20(a).

Después de haber eliminado elementos de la cadena en ciertas posiciones de alguno de los ejes (particularmente se está haciendo sobre el *Eje 0-2*) se puede pensar que un elemento a_i de la cadena ya escalada tiene valor de posición absoluta en el *Eje 1-3* igual que la del elemento a_{i+1} que lo sucede. Este caso ideal garantizaría que el contorno que representa la cadena permanezca siendo continuo.

Sin embargo, esto no siempre ocurre y existen algunos casos adicionales que se deben manejar. Igual que en el Algoritmo 3, cuando la continuidad no se logra se insertan elementos en la cadena para que vuelvan a hacer continuo el contorno.

Se detectan 8 posibles casos adicionales al caso ideal que están descritos en el Cuadro 2.4 y ejemplificados en la Figura 2.20(b). Además se muestra el algoritmo que resuelve cada uno de esos casos (Algoritmos 5-12) para después ser integrados en el algoritmo de escalamiento por un factor $r < 1$ (Algoritmo 4).

Caso	Algoritmo
0	IDEAL
1	ARRIBA-DERECHA-ARRIBA (Algoritmo 5)
2	ARRIBA-IZQUIERDA-ARRIBA (Algoritmo 6)
3	ARRIBA-DERECHA-ABAJO (Algoritmo 7)
4	ARRIBA-IZQUIERDA-ABAJO (Algoritmo 8)
5	ABAJO-DERECHA-ABAJO (Algoritmo 9)
6	ABAJO-IZQUIERDA-ABAJO (Algoritmo 10)
7	ABAJO-DERECHA-ARRIBA (Algoritmo 11)
8	ABAJO-IZQUIERDA-ARRIBA (Algoritmo 12)

Cuadro 2.4: Casos en el proceso de escalamiento $ty < 1$

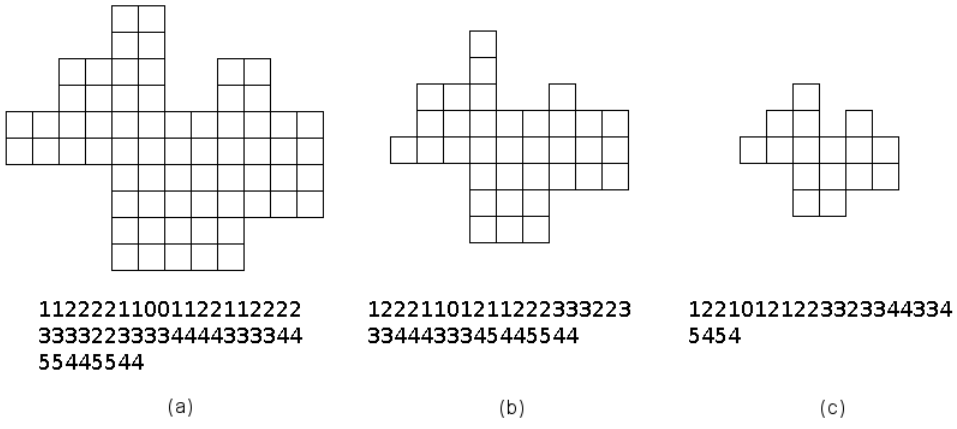


Figura 2.21: Escalamiento en factores r_x y r_y igual a: (a) 1.0; (b) 0.75; (c)0.5

Mientras el caso 0 es el ideal, el caso 1 por ejemplo, se distingue porque un elemento a_i de la cadena ya escalada, lleva la dirección de un elemento *0* en *SACC*, y el elemento a_{i+1} mantiene esa misma dirección ascendente en el *Eje 0-2* pero $pos^*_{Eje1-3}(a_i) < pos^*_{Eje1-3}(a_{i+1})$ por lo que se denota ese caso como *ARRIBA-DERECHA-ARRIBA*. Figura 2.20(b). De este modo pueden ser analizados los otros siete casos.

La Figura 2.21 muestra formas discretas escaladas por el algoritmo de decremento y dando resultados aproximados al de una interpolación de orden cero.

Algoritmo 4 Algoritmo para escalar un código SACC en un factor $r_y < 1$

```
SCALE_DOWN(A, length, S, slength)
   S[0] ← A[0]
   lastVal ← A[0]
   idx ← 1
   lastX ← pos*Eje1−3 (A[0])
   lastY ← pos*Eje0−2 (A[0])
   for j ← 1 to length do
      i ← Mod (j, length)
      if pos*Eje1−3(A[i]) = Round(Round(pos*Eje1−3(A[i]) * ry)/ry)
         or pos*Eje1−3 (A[i]) = Height (A) then
         nextVal ← Mod (Mod (lastVal, 4) + 4, 4))
         switch (nextVal)
            case 0    %Va hacía arriba
               Algoritmo 5
               Algoritmo 6
               Algoritmo 7
               Algoritmo 8
            case 2    %Va hacía abajo
               Algoritmo 9
               Algoritmo 10
               Algoritmo 11
               Algoritmo 12
         S[idx] ← A[i]
         lastVal ← A[i]
         lastX ← pos*Eje1−3 (A[i])
         lastY ← pos*Eje0−2 (A[i])
         idx ← idx + 1
```

Algoritmo 5 Caso ARRIBA-DERECHA-ARRIBA de escalamiento $ry < 1$

```
if pos*_{Eje1-3} (A[i]) > lastX and pos*_{Eje0-2} (A[i]) > lastY then
   if nextVal = 2 or nextVal = 3 then
      NextIteration
   while i ≥ 2 and lastX ≠ pos*_{Eje1-3} (A[i]) and
      Mod (Mod (S[idx − 2], 4) + 4, 4) = 3 do
      S[idx − 2] ← S[idx − 1]
      idx ← idx − 1
      lastX ← lastX + 1
   if lastX ≠ pos*_{Eje1-3} (A[i]) then
      S[idx − 1] ← S[idx − 1] + 1
      lastX ← lastX + 1
      while pos*_{Eje1-3} (A[i]) > lastX do
         S[idx] ← S[idx − 1]
         idx ← idx + 1
         lastX ← lastX + 1
      S[idx] ← S[idx − 1] − 1
      idx ← idx + 1
```

Algoritmo 6 Caso ARRIBA-IZQUIERDA-ARRIBA de escalamiento $ry < 1$

```
if pos*_{Eje1-3} (A[i]) < lastX and pos*_{Eje0-2} (A[i]) > last then
   if nextVal = 1 or nextVal = 2 then
      NextIteration
   while i ≥ 2 and lastX ≠ pos*_{Eje1-3}(A[i]) and
      Mod (Mod (S[idx − 2], 4) + 4, 4) = 1 do
      S[idx − 2] ← S[idx − 1]
      idx ← idx − 1
      lastX ← lastX − 1
   if lastX ≠ pos*_{Eje1-3} (A[i]) then
      S[idx − 1] ← S[idx − 1] − 1
      lastX ← lastX − 1
      while pos*_{Eje1-3} (A[i]) < lastX do
         S[idx] ← S[idx − 1]
         idx ← idx + 1
         lastX ← lastX − 1
      S[idx] ← S[idx − 1] + 1
      idx ← idx + 1
```

Algoritmo 7 Caso ARRIBA-DERECHA-ABAJO de escalamiento $ry < 1$

```
if pos*_Eje1−3 (A[i]) > lastX And pos*_Eje0−2 (A[i]) = lastY then
   if nextVal = 0 or nextVal = 3 then
      NextIteration
   while i ≥ 2 and lastX ≠ pos*_Eje1−3 (A[i]) and
      Mod (Mod (S[idx − 2], 4) + 4, 4) = 3 do
      S[idx − 2] ← S[idx − 1]
      idx ← idx − 1
      lastX ← lastX + 1
   if lastX ≠ pos*_Eje1−3 (A[i]) then
      S[idx − 1] ← S[idx − 1] + 1
      lastX ← lastX + 1
      while pos*_Eje1−3 (A[i]) > lastX do
         S[idx] ← S[idx − 1]
         idx ← idx + 1
         lastX ← lastX + 1
```

Algoritmo 8 Caso ARRIBA-IZQUIERDA-ABAJO de escalamiento $ry < 1$

```
if pos*_Eje1−3 (A[i]) < lastX and pos*_Eje0−2 (A[i]) = lastY then
   if nextVal = 0 or nextVal = 1
      NextIteration
   while i ≥ 2 and lastX ≠ pos*_Eje1−3 (A[i]) and
      Mod (Mod (S[idx − 2], 4) + 4, 4) = 1 do
      S[idx − 2] ← S[idx − 1]
      idx ← idx − 1
      lastX ← lastX − 1
   if lastX ≠ pos*_Eje1−3 (A[i]) then
      S[idx − 1] ← S[idx − 1] − 1
      lastX ← lastX − 1
      while pos*_Eje1−3 (A[i]) < lastX do
         S[idx] ← S[idx − 1]
         idx ← idx + 1
         lastX ← lastX − 1
```

Algoritmo 9 Caso ABAJO-DERECHA-ABAJO de escalamiento $ry < 1$

```
if pos*_{Eje1-3} (A[i]) > lastX and pos*_{Eje0-2} (A[i]) < lastY then
   if nextVal = 0 or nextVal = 3 then
      NextIteration
   while i ≥ 2 and lastX ≠ pos*_{Eje1-3} (A[i]) and
      Mod (Mod (S[idx − 2], 4) + 4, 4) = 3 do
      S[idx − 2] ← S[idx − 1]
      idx ← idx − 1
      lastX ← lastX + 1
   if lastX ≠ pos*_{Eje1-3} (A[i]) then
      S[idx − 1] ← S[idx − 1] − 1
      lastX ← lastX + 1
      while pos*_{Eje1-3} (A[i]) > lastX do
         S[idx] ← S[idx − 1]
         idx ← idx + 1
         lastX ← lastX + 1
      S[idx] ← S[idx − 1] + 1
      idx ← idx + 1
```

Algoritmo 10 Caso ABAJO-IZQUIERDA-ABAJO de escalamiento $ry < 1$

```
if pos*_{Eje1-3} (A[i]) < lastX and pos*_{Eje0-2} (A[i]) < lastY then
   if nextVal = 1 or nextVal = 0 then
      NextIteration
   while i ≥ 2 and lastX ≠ pos*_{Eje1-3} (A[i]) and
      Mod (Mod (S[idx − 2], 4) + 4, 4) = 1 do
      S[idx − 2] ← S[idx − 1]
      idx ← idx − 1
      lastX ← lastX − 1
   if lastX ≠ pos*_{Eje1-3} (A[i]) then
      S[idx − 1] ← S[idx − 1] + 1
      lastX ← lastX − 1
      while pos*_{Eje1-3} (A[i]) < lastX do
         S[idx] ← S[idx − 1]
         idx ← idx + 1
         lastX ← lastX − 1
      S[idx] ← S[idx − 1] − 1
      idx ← idx + 1
```

Algoritmo 11 Caso ABAJO-DERECHA-ARRIBA de escalamiento $ry < 1$

```
if pos*_Eje1−3 (A[i]) > lastX And pos*_Eje0−2 (A[i]) = lastY then
   if nextVal = 2 or nextVal = 3 then
      NextIteration
   while i ≥ 2 and lastX ≠ pos*_Eje1−3 (A[i]) and
      Mod (Mod (S[idx − 2], 4) + 4, 4) = 3 do
      S[idx − 2] ← S[idx − 1]
      idx ← idx − 1
      lastX ← lastX + 1
   if lastX ≠ pos*_Eje1−3 (A[i]) then
      S[idx − 1] ← S[idx − 1] − 1
      lastX ← lastX + 1
      while pos*_Eje1−3 (A[i]) > lastX do
         S[idx] ← S[idx − 1]
         idx ← idx + 1
         lastX ← lastX + 1
```

Algoritmo 12 Caso ABAJO-IZQUIERDA-ARRIBA de escalamiento $ry < 1$

```
if pos*_Eje1−3 (A[i]) < lastX and pos*_Eje0−2 (A[i]) = lastY then
   if nextVal = 1 or nextVal = 2 then
      NextIteration
   while i ≥ 2 and lastX ≠ pos*_Eje1−3 (A[i]) and
      Mod (Mod (S[idx − 2], 4) + 4, 4) = 1
      S[idx − 2] ← S[idx − 1]
      idx ← idx − 1
      lastX ← lastX − 1
   if lastX ≠ pos*_Eje1−3 (A[i]) then
      S[idx − 1] ← S[idx − 1] + 1
      lastX ← lastX − 1
      while pos*_Eje1−3 (A[i]) < lastX do
         S[idx] ← S[idx − 1]
         idx ← idx + 1
         lastX ← lastX ∗ 1
```

Capítulo 3

Algoritmos de semejanza entre cadenas

Uno de los principales retos planteados en este trabajo es el de entregar una medida que determine que tan parecidas o en otro caso que tan distintas son dos formas discretas representadas por su código de cadena. El cálculo de esta medida debe de trabajar directamente sobre el mundo unidimensional de una cadena y evitar resolver el problema con los métodos tradicionales en el plano cartesiano. En esta sección se presentan las alternativas para resolver este problema que después serán probadas y comparadas entre sí.

La primera de estas aproximaciones es la del cálculo de la subsecuencia común más larga (*longest common subsequence, LCS*) entre dos cadenas considerando la longitud de dicha subsecuencia como una medida de similitud. Más adelante también será claro que el problema de la *LCS* es solo un caso particular del algoritmo de similitud global entre cadenas [11].

Posteriormente, se estará trabajando con algoritmos con la misma naturaleza de construcción que la de *LCS* y que dan por resultado una distancia entre cadenas. La distancia puede medir la cantidad de cambios a hacer o incluso puede ponderar cada uno de esos cambios y asociar un costo a cada operación, en cuyo caso el problema

se convierte en tratar de reducir el costo total para igualar las cadenas.

Finalmente, se presentan algoritmos que arrojan una medida de similitud entre cadenas y que en realidad están tratando de optimizar el alineamiento de cadenas. En este tipo de algoritmos se puede encontrar la medida de similitud a partir de un alineamiento global, local o incluso un híbrido global-local entre las cadenas. A este último tipo de alineamiento también se le conoce como alineamiento híbrido o semiglobal [13].

Cualquiera que sea la aproximación para resolver el problema, se está entregando un valor que determina la magnitud de la semejanza o desemejanza entre códigos de cadena y dicha magnitud se puede considerar una medida de similitud o de distancia. La similitud y la distancia son dos formas de comparación de cadenas comúnmente confundidas y, sin embargo, existen diferencias importantes entre ellas.

Cuando se trabaja con similitud, se intenta lograr el mejor alineamiento posible entre dos cadenas y obtener una puntuación que defina qué tanto esas cadenas se parecen entre sí. Por el contrario, en el caso de la distancia se asignan costos a las operaciones básicas de edición y se busca un alineamiento en el que la composición de dichas operaciones tenga el menor costo para convertir una cadena en la otra. Es decir, contrario a la similitud, la distancia es la medida que dice que tanto difieren dos cadenas y por eso también se le conoce como medida de disimilitud o desemejanza.

3.1. Subsecuencia común más larga (LCS)

El algoritmo de subsecuencia común más larga (*longest common subsequence, LCS*) [7] es muy usado en aplicaciones biológicas de comparación de cadenas de ADN entre dos distintos organismos. Una banda de ADN se compone de cuatro moléculas llamadas bases que son adenina, citosina, guanina y timina abreviadas con las letras A, C, G y T respectivamente y que se repiten de manera no ordenada en una cadena. Por ejemplo, el ADN de un organismo puede ser una cadena $X = ACCGGTCGAGTGCGCGGAAGCCGGCCGAA$, mientras que el de otro

organismo puede ser $Y = GTCGTTCGGAATGCCGTTGCTCTGTAAA$, y llega a ser común el querer compararlas para obtener una medida de su similitud.

Una manera de decir que las dos cadenas se parecen entre sí es viendo si una de ellas es una subcadena de la otra, o en su defecto, si hay una subcadena de longitud lo suficientemente grande de una cadena que también se presente en la otra cadena. En el ejemplo que se plantea del ADN, ni X ni Y suelen ser subcadenas de la otra. Otra aproximación para decir que dos cadenas son parecidas entre sí es verificar si el número de cambios necesarios para convertir una cadena en la otra es lo suficientemente pequeño, y este intento se hace con la distacia de edición.

Por ahora, para medir la similitud entre las cadena X y Y se encontrará una tercera cadena Z en la que se incluyen los elementos que van apareciendo tanto en X como en Y de modo ascendente, aunque no necesariamente de modo consecutivos. Por supuesto, mientras más grande sea Z, más similares serán X y Y. En el ejemplo del ADN, la subsecuencia común más larga que existe entre X y Y es $Z = GTCGTCGGAAGCCGGCCGAA$.

Formalizando un poco, dada una secuencia $X = x_1x_2...x_{n_X}$, se dice que otra secuencia $Z = z_1z_2...z_k$ es una subsecuencia de X si existe una secuencia estrictamente incremental $I = i_1i_2...i_k$ de índices de X tales que para toda $j = 1, 2, ..., k$, se tiene que $x_{i_j} = z_j$. Por ejemplo, $Z = BCDB$ es una subsecuencia de $X = ABCBDAB$ con su correspondiente secuencia de índices $I = 2\ 3\ 5\ 7$.

Dadas dos secuencias X y Y, se dice que Z es una subsecuencia común de X y Y si Z es una subsecuencia de X y también lo es de Y. Por ejemplo, si $X = ABCBDAB$ y $Y = BDCABA$, la secuencia BCA es una subsecuencia común de X y Y. Sin embargo, BCA de longitud 3 no es la subsecuencia común más larga. Tanto $BCBA$ como $BDAB$ son LCS de X y Y ya que ambas tienen longitud 4 y no existen subcadenas comunes entre X y Y de longitud 5 o superior.

En el problema de la subsecuencia común más larga se tienen dos secuencias $X = x_1x_2...x_{n_X}$ y $Y = y_1y_2...y_{n_Y}$ y se desea encontrar la longitud máxima para una subsecuencia común de X y Y. Este problema puede ser resuelto de forma eficiente

usando Programación Dinámica [7].

En el algoritmo para la solución de este problema se define una matriz D de dimensiones $(n_X + 1) \times (n_Y + 1)$ que tendrá en su elemento $D[i, j]$ la longitud de la LCS de las secuencias $X_i = x_1...x_i$ y $Y_j = y_1...y_j$. Si $i = 0$ o $j = 0$, una de las secuencias tiene longitud 0 por lo que que la LCS tendrá longitud 0. Una subestructura para la solución del problema da la fórmula:

$$D(i,j) = \begin{cases} 0 & i = 0 \ \vee \ j = 0 \\ D(i-1, j-1) + 1 & i, j > 0 \ \wedge \ x_i = y_j \\ max(D(i, j-1), D(i, j-1)) & i, j > 0 \ \wedge \ x_i \neq y_j \end{cases} \tag{3.1}$$

Basado en la ecuación anterior, el Algoritmo 13 implementa el cálculo de la matriz D apartir de dos secuencias $X = x_1x_2...x_{n_X}$ y $Y = y_1y_2...y_{n_Y}$. El algoritmo también puede manejar una matriz T de dimensiones $(n_X + 1) \times (n_Y + 1)$ paralela a D por si fuera deseado saber la ruta por al que se consiguió la LCS.

La Figura 3.1 muestra las matrices generadas por el Algoritmo 13 con las secuencias $X = ABCBDAB$ y $Y = BDCABA$. La longitud de la LCS queda en $D[m, n]$ que para este caso es 4. Si se quiere saber que elementos forman la LCS se puede hacer una algoritmo recursivo que vaya siguiendo en reversa la ruta trazada por los elementos de T. El que exista un elemento $T[i, j] =$ " $\nwarrow$ " en esa ruta, significa que el elemento x_i y el elemento y_j son iguales y que su valor se debe incluir como un elemento de la LCS. Para el ejemplo de la Figura 3.1 la $LCS = BCBA$.

El tiempo de ejecución del algoritmo es $O(n_X n_Y)$ puesto que se recorre una vez cada elemento de la matriz y por cada elemento el tiempo de cómputo es $O(1)$. En espacio de almacenamiento el algoritmo así propuesto también requiere de espacio $O(n_X n_Y)$, pero con algunas observaciones, fácilmente puede ser reducido a $O(2n_X)$ ó $O(2n_Y)$ puesto que lo que interesa conocer es la longitud de la LCS y para ello la matriz T no es requerida. Las únicas filas de D que importan son la que está siendo procesada y la fila inmediata anterior.

Algoritmo 13 Cálculo de la Subsecuencia Común Más Larga (LCS)

```
LCS(X, Y)
   nx ← Length(X)
   ny ← Length(Y)
   for i ← 0 to nx do
      D[i,0] ← 0
   for j ← 1 to ny do
      D[0,j] ← 0
   for i ← 1 to nx do
      for j ← 1 to ny do
         if X[i] = Y[i] then
            D[i,j] ← D[i−1,j−1] + 1
            T[i,j] ← "↖"
         else
            if D[i-1,j] ≥ D[i,j−1] then
               D[i,j] ← D[i−1,j]
               T[i,j] ← "↑"
            else
               D[i,j] ← D[i,j−1]
               T[i,j] ← "←"
   return D and T
```

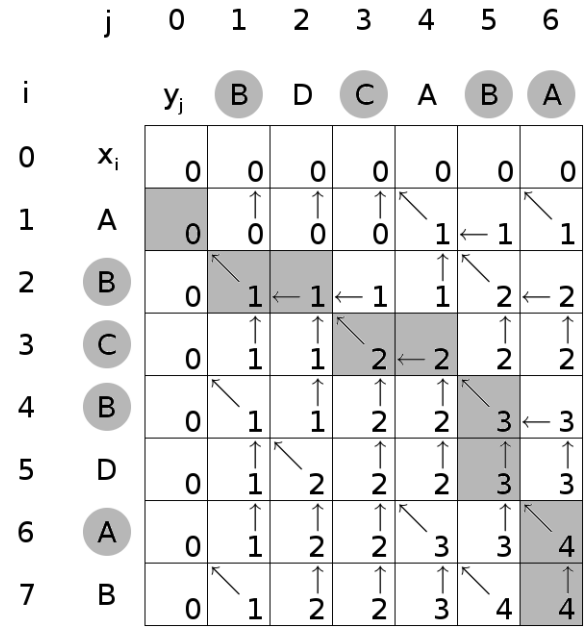


Figura 3.1: Representación de las matrices D y T calculadas con el Algoritmo 13

3.2. Distancia entre cadenas

Se puede decir que dadas dos cadenas, saber si son iguales es una tarea bastante simple. No importa qué representan dichas cadenas, ni en el contexto en el que éstas se presentan, basta simplemente con comparar uno a uno todos sus elementos de manera ordenada y decidir a partir de la igualdad de todos ellos si las cadenas son iguales o diferentes. Basta que exista un elemento en la i-ésima posición de una cadena que no sea igual al elemento en la misma posición de la otra cadena para concluir que se trata de cadenas distintas. Lo mismo ocurre con las diferencias de una cadena por exceso u omisión de elementos con respecto a otra: si las cadenas no tienen la misma longitud se puede concluir que son distintas.

Sin embargo, este escenario no siempre es suficiente. Hay ocasiones en las que dos cadenas no son iguales pero sí son parecidas, lo cual suele ser información importante para tomar algunas decisiones sobre ellas, pero al mismo tiempo convierte el problema en otro más díficil de resolver. En este nuevo problema ya puede llegar a ser exigencia el conocimiento de lo que representan esas cadenas, de su naturaleza y de que factores son importantes para decidir que tan parecidas son entre sí.

El problema en su forma más general, es ser capaz de decidir que dos cadenas se parecen permitiendo un número de "errores" en su emparejamiento. Por supuesto, habrá formas de emparejar las cadenas que sean más convenientes que otras, y entonces, encontrar la forma de emparejamiento óptima será encontrar aquel emparejamiento en el que el número de errores o el costo que estos tienen es mínimo.

La naturaleza del problema depende en gran medida del tipo de errores que se considera manejar y, en dependencia de ello, las soluciones pueden variar desde algoritmos de orden lineal hasta algoritmos NP-completos [8]. Afortunadamente, para la naturaleza de los problemas planteados en este trabajo, basta con considerar aquellos algoritmos definidos en términos de variación de algunos elementos de las cadenas y de la modificación del costo total que se tiene por hacer esas variaciones.

Visto de este modo, parece más claro que el camino a seguir es el de mínimizar el

costo total resultado de convertir una cadena en otra, entendiendo que en cada uno de los errores de emparejamiento entre cadenas se tendrá un costo asociado. Desde otra perspectiva, el problema se puede observar en términos de la energía mínima que se requiere para convertir una cadena en otra, teniendo que cada cambio realizado en una cadena demanda energía para ser hecho. Sea cual sea el punto de vista, en este caso se está calculando una distancia entre las dos cadenas que mientras menor sea más similares serán entre ellas. La distancia aplicada a un conjunto U es una función $D : U \times U \mapsto \mathcal{R}$ tal que

1. $D(x, x) = 0$ para toda $x \epsilon U$ y $D(x, y) > 0$ para $x \neq y$ (distancia 0 hacía sí mismo)

2. $D(x, y) = D(y, x)$ para toda $x, y \epsilon U$ (medida de distancia simétrica)

3. $D(x, z) \leq D(x, y) + D(y, z)$ para toda $x, y, z \epsilon U$ (desigualdad del triángulo)

Es importante notar algunas de las restricciones implícitas en estas reglas. Por ejemplo, la primera regla muestra que ningún costo asociado a una operación de edición puede tener un valor negativo ya que podría ocasionar que el valor de distancia total mínimo no tuviera sentido. Para la segunda regla es importante notar la simetría que existe en la distancia entre cadenas. Otra restricción se encuentra en la tercera regla donde la desigualdad del triángulo se logra combinando las dos reglas anteriores, o de otro modo ésta no se cumpliría. A partir de estas reglas también se puede observar que la única forma para tener una distancia total mínima de 0 es comparando dos cadenas iguales.

Uno de los casos mejor estudiados para problemas de emparejamiento de cadenas (*string matching*) permitiendo errores se conoce como distancia de edición (*edit distance*), el cual nos permite eliminar, insertar o remplazar elementos en cualquiera de las dos cadenas [8]. Si operaciones distintas tienen costos distintos, o incluso, si los costos asociados dependen de los caracteres que están siendo evaluados, se está hablando de la distancia de edición general (*general edit distance*). De otro modo,

si todas las operaciones y costos asociados a ellas es 1 independientemente de los caracteres evaluados, se está hablando de distancia de edición simple o simplemente de distancia de edición.

Para este último caso se entiende entonces que la distancia de edición es el mínimo número de mutaciones puntuales requeridas para convertir una cadena en otra y donde una mutación puntual puede ser la inserción, la eliminación o el cambio de un elemento de la cadena.

3.2.1. Cálculo de la distancia de edición simple

El algoritmo de Levenshtein es un algoritmo de Programación Dinámica usado para calcular la distancia de edición simple o también conocida como distancia de Levenshtein en reconocimiento al artículo del mismo V.I. Levenshtein [9] en donde por primera vez se trata el problema de la distancia de edición de manera formal.

Aunque la definición de distancia de edición implica que todas las operaciones son hechas sobre una de las cadenas con el fin de convertirla en la otra cadena, en ocasiones es más conveniente pensar la defición como el mínimo número de operaciones aplicadas a ambas cadenas con el fin de transformarlas en una tercer cadena común. Este nuevo punto de vista sigue siendo consistente con la definición debido a que la inserción de un elemento en una cadena puede ser vista como la eliminación de un elemento en la otra cadena y viceversa [10].

Para dos cadenas X y Y de longitudes n_X y n_Y respectivamente, $D(i, j)$ será definida como la distancia de edición entre $X[1...i]$ y $Y[1...j]$. Esto nos sugiere que para el caso particular $D(n_X, n_Y)$ se obtiene la distancia de edición entre X y Y.

En general el paradigma de programación dinámica siempre tiene un componente de recurrencia que va estableciendo una relación recursiva entre el valor $D(i, j)$ y los valores de D para un par de índices menores a i y j. Para $D(i, j)$ donde ya no existen índices más pequeños de i o de j se deben definir de manera explícita los llamados casos base o condiciones base. En el problema de la distancia de edición,

estas condiciones base se establecen cuando i o j son 0 y quedan definidas como:

$$D(i,0) = i$$

y

$$D(0,j) = j$$

La condición base $D(i,0) = i$ es claramente correcta porque la única forma de transformar los primeros i caracteres de X en 0 caracteres de Y es borrando cada uno de esos i caracteres. Análogamente, la condición base $D(0,j) = j$ es correcta por la misma razón.

La relación de recurrencia para $D(i,j)$ con valores de i y j mayores de 0 es:

$$D(i,j) = Min(D(i-1,j-1) + t(i,j), D(i-1,j) + 1, D(i,j-1) + 1)$$

donde $t(i,j) = 0$ si $X[i] = Y[j]$ y $t(i,j) = 1$ en otro caso. La correctez del algoritmo puede ser probada [10] pero dicha prueba no se incluye en este trabajo. Las relaciones de recurrencia entonces obtenidas para el cálculo de la distancia de edición se pueden describir de la forma:

$$D(i,j) = \begin{cases} j & i = 0 \\ i & j = 0 \\ Min \begin{cases} D(i-1,j)+1, \\ D(i,j-1)+1, \\ D(i-1,j-1) \end{cases} & i,j > 0 \ \wedge \ X[i] = Y[j] \\ Min \begin{cases} D(i-1,j)+1, \\ D(i,j-1)+1, \\ D(i-1,j-1)+1 \end{cases} & i,j > 0 \ \wedge \ X[i] \neq Y[j] \end{cases} \tag{3.2}$$

Otro componente importante del paradigma de programación dinámica aplicado al problema de la distancia de edición es el de usar las relaciones de recurrencia anteriormente obtenidas para el cómputo eficiente del valor $D(n_X, n_Y)$. Una aproximación ingenua a la solución del problema sería codificar las relaciones de recursión para que dadas las entradas n_X y n_Y en la función D de distancia se calcule $D(n_X, n_Y)$ haciendo las llamadas recursivas necesarias dentro de la función hasta llegar a las condiciones base. Esta aproximación, también conocida como *top-down* es extremadamente ineficiente para valores muy grandes de n_X y n_Y puesto que el número de llamadas a D va creciendo exponecialmente conforme n_X y n_Y crecen.

Una implementación más eficiente y muy simple consiste en darse cuenta que dado que solo hay $(n_X + 1) \times (n_Y + 1)$ combinaciones de i y de j, nada más hay que calcular ese número de llamadas a D e ir reutilizando de manera eficiente las llamadas anteriores. A esta aproximación se le conoce como *bottom-up*, e igual que LCS ocupa una matriz justamente de $(n_X + 1) \times (n_Y + 1)$ para ir almacenando las llamadas a la función D. A partir de este momento se hará referencia a esa matriz también como D y la forma de identificarla respecto a la función de distancia la dará el contexto sabiendo que existe una equivalencia en ambos casos ya que la matriz es

Algoritmo 14 Cálculo de la distancia de edición

```
EDIT_DISTANCE(X, Y)
   nx ← length(X)
   ny ← length(Y)
   for i ← 0 to nx do
      D[i, 0] ← i
   for j ← 1 to ny do
      D[0, j] ← j
   for i ← 1 to nx do
      for j ← 1 to ny do
         if X[i] = Y[i] then
            costCopying ← D[i-1, j-1]
         else
            costCopying ← D[i-1, j-1] + 1
         costInserting ← D[i, j-1] + 1
         costDeleting ← D[i-1, j] + 1
         D[i, j] ← Min(costCopying,costInserting,costDeleting)
   return D[nx, ny]
```

una representación de las salidas de la función.

En esta aproximación, primero se computa $D(i, j)$ para los valores más pequeños posibles de i y j y se almacena ese resultado justamente en la posición i y j de la matriz D. La matriz se sigue llenando en orden estrictamente creciente para los valores de j y cuando se llega al final de las columnas para cierto valor de i, se procede nuevamente con los cálculos desde la columna $j = 0$ sobre la fila $i + 1$. Otra manera en que se puede recorrer la matriz es recorriendo primero las posiciones de i y cuando éstas son agotadas se inicia nuevamente con valores de fila $i = 0$ pero ahora en la columna $j + 1$. Es importante notar que este orden de recorrido es así porque las relaciones de recurrencia requieren para el cálculo de $D(i, j)$ que las distancias $D(i - 1, j)$, $D(i, j - 1)$ y $D(i - 1, j - 1)$ ya hallan sido computadas y cualquiera de los dos recorridos anteriormente planteados satisfacen esa condición. El Algoritmo 14 funciona con el primer recorrido para calcular la distancia de edición.

Parecido a lo que ocurre con el cálculo de la *LCS*, después de haber llenado la

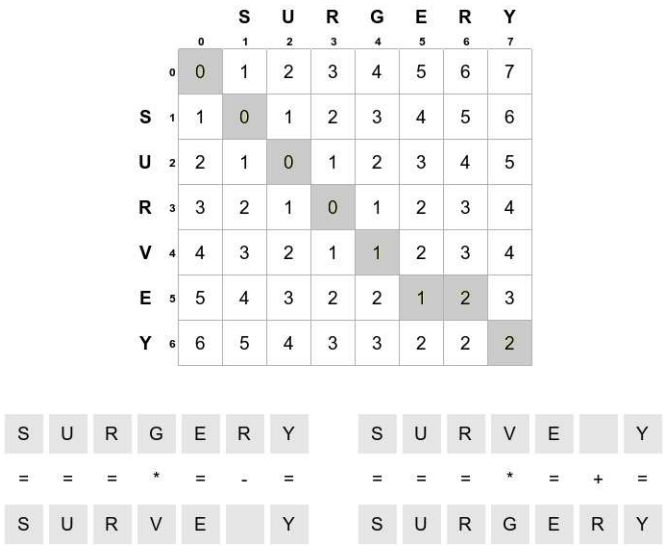


Figura 3.2: Matriz D resultado del algoritmo de distancia de edición simple con su alineamiento óptimo ($Distancia = 2$)

matriz de distancias D, el resultado de la distancia de edición entre las cadenas X y Y estará en $D[n_X,n_Y]$. La Figura 3.2 nos muestra la matriz D resultado del cálculo de la distancia de edición simple entre dos cadenas cuya distancia de edición se muestra ser 2. También se muestra el alineamiento óptimo que existe entre las dos cadenas involucradas denotando entre ellas la operación de edición necesaria (Emparejamiento '=', Sustitución '*', Inserción '+', Eliminación '-').

Respecto al análisis del tiempo de ejecución del algoritmo, cuando se computa el valor de $D(i, j)$ para cualquier i y j podemos considerar que toma tiempo constante hacer un par de comparaciones y de operaciones aritméticas. Si tomamos en cuenta que hay $O(n_X n_Y)$ distancias que calcular, sabremos entonces que ese también es el tiempo que toma calcular la distancia de edición simple y que al igual que en el cálculo de la LCS, con algunas consideraciones es claro que el espacio requerido puede ser reducido de $O(n_X n_Y)$ a $O(2n_X)$ ó $O(2n_Y)$.

3.2.2. Cálculo de la distancia de edición general (distancia de edición ponderada)

Una generalización interesante y simple de realizar es la de asociar costos arbitrarios con cada operación de edición. Cualquier eliminación tiene un costo asociado que usualmente es el mismo que el de una inserción y en este trabajo no distinguiremos entre ellos usando g (*gap*); una sustitución tendrá un costo m (*mismatch*) y un buen emparejamiento un costo M (*match*) que usualmente será 0 cuando de distancia se trate. Estos costos pueden ser determinados de manera fija, o pueden ser calculados en función de los elementos de las dos cadenas que están siendo operados.

Con costos arbitrarios asignados, el problema del cálculo de la distancia de edición general es encontrar el costo total mínimo de operaciones que conviertan la cadena X y Y en una tercera cadena común. Es evidente que entonces la distancia de edición simple es tan solo un caso particular de la distancia de edición general o también conocida como distancia de edición ponderada con $g = m = 1$ y $M = 0$.

Al igual que con la distancia de edición simple, el problema de la distancia de edición general puede ser resuelto en tiempo $O(n_X n_Y)$ y $D(i, j)$ ahora representa el costo total de operaciones de edición mínimo entre las cadenas $X[1...i]$ y $Y[1...j]$. Se puede seguir usando la misma función $t(i, j)$ para el manejo de las operaciones de sustitución e igualdad con la diferencia que ahora $t(i, j) = M$ si $X[i] = Y[j]$ y $t(i, j) = m$ en otro caso. Es importante recordar que para el caso del cálculo de la distancia de edición se esperan parámetros del tipo $M = 0$, $m > 0$ y $g > 0$ ya que de otro modo la distancia dejaría de tener sentido en lo que representa. Ahora nuestras condiciones base son:

$$D(i, 0) = i * g$$

y

$$D(0, j) = j * g$$

La recurrencia general es:

$$D(i,j) = Min(D(i-1,j-1) + t(i,j), D(i-1,j) + g, D(i,j-1) + g)$$

Las relaciones de recurrencia entonces obtenidas para el cálculo de la distancia de edición general se pueden describir de la forma:

$$D(i,j) = \begin{cases} j * g & i = 0 \\ i * g & j = 0 \\ Min \begin{cases} D(i-1,j) + g, \\ D(i,j-1) + g, \\ D(i-1,j-1) + M \end{cases} & i,j > 0 \ \wedge \ X[i] = Y[j] \\ Min \begin{cases} D(i-1,j) + g, \\ D(i,j-1) + g, \\ D(i-1,j-1) + m \end{cases} & i,j > 0 \ \wedge \ X[i] \neq Y[j] \end{cases} \tag{3.3}$$

El Algoritmo 15 muestra una implementación del cálculo de la distancia de edición general [11].

La Figura 3.3 muestra la matriz D resultado del cálculo de la distancia de edición general entre dos cadenas cuyo costo mínimo de operaciones de edición es 7 considerando los costos $M = 0$, $m = 3$ y $g = 4$. También se muestra el alineamiento óptimo que existe entre las dos cadenas involucradas denotando entre ellas la operación de edición necesaria para convertir la primera cadena en la segunda (Emparejamiento '=', Sustitución '*', Inserción '+', Eliminación '-').

Algoritmo 15 Cálculo de la distancia de edición general

```
GENERAL_EDIT_DISTANCE(X, Y)
   nx ← length(X)
   ny ← length(Y)
   for i ← 0 to nx do
      D[i, 0] ← i * g
   for j ← 1 to ny do
      D[0, j] ← j * g
   for i ← 1 to nx do
      for j ← 1 to ny do
         if X[i] = Y[i] then
            costCopying ← D[i-1, j-1] + M
         else
            costCopying ← D[i-1, j-1] + m
         costInserting ← D[i, j-1] + g
         costDeleting ← D[i-1, j] + g
         D[i, j] ← Min(costCopying,costInserting,costDeleting)
   return D[nx, ny]
```

		S	U	R	G	E	R	Y
	0	1	2	3	4	5	6	7
0	0	4	8	12	16	20	24	28
S 1	4	0	4	8	12	16	20	24
U 2	8	4	0	4	8	12	16	20
R 3	12	8	4	0	4	8	12	16
V 4	16	12	8	4	3	7	11	15
E 5	20	16	12	8	7	3	7	11
Y 6	24	20	16	12	11	7	6	7

S	U	R	G	E	R	Y	
=	=	=	*	=	-	=	
S	U	R	V	E		Y	

S	U	R	V	E		Y	
=	=	=	*	=	+	=	
S	U	R	G	E	R	Y	

Figura 3.3: Matriz D resultado del algoritmo de distancia de edición general con su alineamiento óptimo ($Distancia = 7$)

3.3. Similitud entre cadenas

A continuación se presenta otra forma de seguir comparando dos secuencias obteniendo como resultado una medida de semejanza. La meta para estos algoritmos de comparación será la de encontrar el mejor alineamiento entre dos cadenas teniendo en cuenta que, a un mejor alineamiento corresponderá una mayor medida de similitud. Existen principalmente tres tipos de alineamiento que son global, local, y también un tercer tipo de alineamiento que es una mezcla de los dos anteriores y se conoce como semiglobal. Es importante mencionar que dado que el problema de la medida de similitud y el del alineamiento global son equivalentes, es posible referirse indistintamente a cualquiera de los dos planteamientos.

Todos los algoritmos se siguen resolviendo de forma eficiente por implementaciones de Programación Dinámica y se sigue la misma línea que en la solución de la distancia de edición. Más aun, se mostrará que la diferencia entre el cálculo de la medida de similitud (global) y de la distancia de edición estriba en la selección de los parámetros de comparación y una sutil modificación del algoritmo de distancia de edición para obtener el máximo en la función objetivo.

3.3.1. Similitud global entre cadenas (alineamiento global)

Como ya se ha comentado, la meta a alcanzar es la obtención de una medida de similitud global mediante un algoritmo eficiente que tome dos secuencias y determine el mejor alineamiento posible entre ellas. Queda por definir qué hace a un alineamiento mejor sobre otro.

Por lo pronto, habrá que definir a un alineamiento como la inserción de espacios en blanco de modo arbitrario en ciertas posiciones de dos cadenas, de tal modo que al final ambas cadenas acaben con la misma cantidad de elementos. Toda vez que ahora las cadenas aumentadas tienen la misma cantidad de elementos, una cadena puede ser puesta sobre la otra creando una correspondencia directa entre los caracteres y espacios de una cadena con los caracteres y espacios de la otra. Es oportuno

$$\begin{array}{cccccccccccccc} G & A & G & C & \_ & G & G & A & A & T & T & G & C & A & C \\ G & A & T & C & A & G & G & A & A & C & T & G & \_ & A & G \end{array}$$

Figura 3.4: Ejemplo de alineamiento global óptimo entre dos cadenas con parámetros $M = 1$, $m = -1$ y $g = -2$ ($Similitud = 3$)

mencionar que dos espacios en blanco no se pueden alinear aunque estos sí pueden ser insertados tanto al inicio como al final de cualquiera de las cadenas.

Dado un alineamiento entre dos secuencias es posible asignar una puntuación total resultado de ir sumando M a cada igualdad entre un elemento de la primera cadena y su correspondiente de la segunda cadena. Igualmente se suma un valor m en caso de que tenga que hacerse el remplazo de un elemento por otro para lograr la igualdad y finalmente un valor g para el caso en el que se encuentre alineado un elemento en una cadena con un espacio en blanco de la otra.

La Figura 3.4 nos muestra un ejemplo del alineamiento global entre dos cadenas con parámetros de comparación $M = 1$, $m = -1$ y $g = -2$ lo cual arroja una puntuación final de 3 debido a 10 elementos iguales, 3 remplazos y 2 eliminaciones. Se dice que el alineamiento es óptimo ya que no existe otro alineamiento entre esas dos cadenas que supere esa puntuación.

Es importante notar que aunque el sistema de puntuación parece muy arbitrario, en realidad no lo es tanto ya que ese sistema es comúnmente usado en la práctica. Más adelante se dedica una explicación completa a la elección de los parámetros de comparación entre secuencias.

La pregunta ahora se centra en cómo se obtuvo ese alineamiento óptimo. Un primer intento ingenuo de solución sería generar todos los posibles alineamientos y revisar cuál de ellos es el de máxima puntuación. De cualquier modo, el número de alineamientos posibles entre dos cadenas es exponencial por lo que esto da como resultado un algoritmo prohibitivamente lento en su ejecución.

Otro intento, como ya se supone, usa Programación Dinámica y por tanto nue-

vamente intenta ir resolviendo los componentes recursivos hasta llegar a un caso base. Es decir, teniendo dos cadenas X y Y, en lugar de intentar determinar la similitud que existe entre ellas de forma directa se hará apartir de la similitud que van teniendo sus prefijos.

Digamos que la longitud de X es n_X y la de Y es n_Y. Existen entonces $n_X + 1$ posibles prefijos de X y $n_Y + 1$ prefijos de Y ya que se puede incluir una cadena vacia como parte de la solución. Entonces, al igual que en el cálculo de la distancia entre cadenas, se requerirá de una matriz S_G de $(n_X + 1) \times (n_Y + 1)$ para ir almacenando los resultados parciales y donde en la entrada $S_G[i, j]$ contiene la similitud entre los prefijos $X[1...i]$ y $Y[1...j]$. Es decir, en el caso de la similitud global S_G además de ser la función de similitud global, es también la matriz que va almacenando los resultados parciales (ésta dualidad entre la función de similitud y su representación en una matriz ya había sido anteriormente descrita para el caso de la distancia de edición y seguirá siendo ocupada para otros tipos de similitud).

En caso de que una de las cadenas que está siendo comparada sea una cadena vacia, se dará como resultado una puntuación derivada de eliminar cada uno de los elementos de la otra cadena. Esto sugiere entonces que los casos base son:

$$S_G(i, 0) = i * g$$

y

$$S_G(0, j) = j * g$$

Mientras que la recurrencia general es

$$S_G(i, j) = Max(S_G(i - 1, j - 1) + t(i, j), S_G(i - 1, j) + g, S_G(i, j - 1) + g)$$

con $t(i, j) = M$ si $X[i] = Y[j]$ y $t(i, j) = m$ en otro caso. Las relaciones de recurrencia

obtenidas para el cálculo del alineamiento global óptimo se pueden describir de la forma:

$$S_G(i,j) = \begin{cases} j * g & i = 0 \\ i * g & j = 0 \\ Max \begin{cases} S_G(i-1,j) + g, \\ S_G(i,j-1) + g, \\ S_G(i-1,j-1) + M \end{cases} & i,j > 0 \ \wedge \ X[i] = Y[j] \\ Max \begin{cases} S_G(i-1,j) + g, \\ S_G(i,j-1) + g, \\ S_G(i-1,j-1) + m \end{cases} & i,j > 0 \ \wedge \ X[i] \neq Y[j] \end{cases} \tag{3.4}$$

Es notorio que la ecuación 3.4 es equivalente a la ecuación 3.3 salvo por el remplazo de la función Min por Max. Es entonces de esperar que el algoritmo para llenar las matrices S_G y D también sea equivalente teniendo la misma diferencia. Algoritmo 16.

La Figura 3.5 muestra la matriz S_G resultado del cálculo de la medida de similitud global entre las dos cadenas de la Figura 3.4. También se muestra que el alineamiento óptimo que existe entre las dos cadenas involucradas es 3 denotando entre ellas la operación de edición necesaria (Emparejamiento '=', Sustitución '*', Inserción '+', Eliminación '-').

El análisis de la eficiencia del algoritmo de alineamiento global mantiene, al igual que el algoritmo de distancia de edición, una cota $O(n_X n_Y)$ puesto que esencialmente la diferencia entre ambos algoritmos está en los parámetros de comparación lo cual no afecta en nada el análisis.

Algoritmo 16 Cálculo de la medida de similitud global (alineamiento global)

```
GLOBAL_SIMILARITY(X, Y)
   nx ← Length(X)
   ny ← Length(Y)
   for i ← 0 to nx do
      S[i,0] ← i * g
   for j ← 1 to ny do
      S[0,j] ← j * g
   for i ← 1 to nx do
      for j ← 1 to ny do
         if X[i] = Y[i] then
            costCopying ← S[i-1, j-1] + M
         else
            costCopying ← S[i-1, j-1] + m
         costInserting ← S[i, j-1] + g
         costDeleting ← S[i-1, j] + g
         S[i,j] ← Max(costCopying,costInserting,costDeleting)
   return S[nx, ny]
```

		G	A	T	C	A	G	G	A	A	C	T	G	A	G
	0	-2	-4	-6	-8	-10	-12	-14	-16	-18	-20	-22	-24	-26	-28
G	-2	1	-1	-3	-5	-7	-9	-11	-13	-15	-17	-19	-21	-23	-25
A	-4	-1	2	0	-2	-4	-6	-8	-10	-12	-14	-16	-18	-20	-22
G	-6	-3	0	1	-1	-3	-3	-5	-7	-9	-11	-13	-15	-17	-19
C	-8	-5	-2	-1	2	0	-2	-4	-6	-8	-8	-10	-12	-14	-16
G	-10	-7	-4	-3	0	1	1	-1	-3	-5	-7	-9	-9	-11	-13
G	-12	-9	-6	-5	-2	-1	2	2	0	-2	-4	-6	-8	-10	-10
A	-14	-11	-8	-7	-4	-1	0	1	3	1	-1	-3	-5	-7	-9
A	-16	-13	-10	-9	-6	-3	-2	-1	2	4	2	0	-2	-4	-6
T	-18	-15	-12	-9	-8	-5	-4	-3	0	2	3	3	1	-1	-3
T	-20	-17	-14	-11	-10	-7	-6	-5	-2	0	1	4	2	0	-2
G	-22	-19	-16	-13	-12	-9	-6	-5	-4	-2	-1	2	5	3	1
C	-24	-21	-18	-15	-12	-11	-8	-7	-6	-4	-1	0	3	4	2
A	-26	-23	-20	-17	-14	-11	-10	-9	-6	-5	-3	-2	1	4	3
C	-28	-25	-22	-19	-16	-13	-12	-11	-8	-7	-4	-4	-1	2	3

G	A	G	C	_	G	G	A	A	T	T	G	C	A	C
=	=	*	=	+	=	=	=	=	*	=	=	-	+	*
G	A	T	C	A	G	G	A	A	C	T	G	_	A	G

Figura 3.5: Matriz S_G resultado del algoritmo de similitud global con su respectivo alineamiento y usando los parámetros $M = 1$, $m = -1$ y $g = -2$ ($Similitud = 3$)

3.3.2. Similitud local entre cadenas (alineamiento local)

En muchas aplicaciones de Biología Molecular, dos cadenas completas pueden no ser altamente similares pero sí contener algunas regiones de alta similitud lo cual puede llegar a ser más importante. Es entonces cuando la tarea de encontrar esas regiones de alta similitud se convierte en el problema central de esas aplicaciones y es el que se conoce como problema de similitud local o alineamiento local.

La definición del problema de alineamiento local entonces se puede describir [12] dadas dos cadenas X y Y, encontrar las subcadenas X^s y Y^s cuya similitud global o alineamiento global es el máximo que puede tener cualquier subcadena de X y Y.

La Figura 3.6(a) muestra el alineamiento local óptimo entre las cadenas alineadas de manera global en la Figura 3.4. Tomando en cuenta los mismos parámetros que se vienen usando, la medida de similitud local obtenida a través de las subcadenas vale 5 producto de 6 emparejamientos y 1 remplazo. A su vez la Figura 3.6(b) muestra los dos alineamientos locales óptimos entre las cadenas $X = PQRAXABCSTVQ$ y $Y = XYAXBACSLL$ usando los parámetros $M = 2$, $m = -2$ y $g = -1$.

En la Figura 3.6(b) las subcadenas $X^s = AXABCS$ y $Y^s = AXBACS$ tienen un valor de similitud global entre sí de 8 y además no existe ningún otro par de subcadenas de X y Y que tenga un valor de similitud global entre sí mayor al de X^s y Y^s. En este caso se dice entonces que el alineamiento local óptimo entre X y Y tiene valor 8 y está definido por las subcadenas $X^s = AXABCS$ y $Y^s = AXBACS$.

$$\begin{array}{ccccccc} G & G & A & A & T & T & G \\ G & G & A & A & C & T & G \end{array}$$

(a)

$$\begin{array}{ccccccc} A & X & \_ & A & B & C & S \\ A & X & B & A & \_ & C & S \end{array} \qquad \begin{array}{ccccccc} A & X & A & B & \_ & C & S \\ A & X & \_ & B & A & C & S \end{array}$$

(b)

Figura 3.6: Alineamiento local óptimo con parámetros $M = 1$, $m = -1$ y $g = -2$: (a) entre las cadenas de la Figura 3.4 ($Similitud = 5$); (b) entre las cadenas $X = PQRAXABCSTVQ$ y $Y = XYAXBACSLL$ ($Similitud = 8$)

Es importante notar que el concepto de *similitud local* entre cadenas o *alineamiento local* queda definido exclusivamente en términos de medida de similitud puesto que su esencia es maximizar una función objetivo, cosa contraria a lo que ocurre en la distancia de edición donde se trata de minimizar. Es por ello que esta sección de alineamiento local no se incluye en el subtema de la distancia de edición.

El alineamiento global entre dos cadenas nos conduce a la obtención de regiones de alta similitud tomando decisiones para terminar alineando de la mejor manera posible toda la cadena X con toda la cadena Y. En este esfuerzo de alinear las cadenas completas es donde regiones de aun mayor similitud suelen perderse en el fin de obtener un óptimo global. Es por esto que para identificar esos altos valores de similitud hay que usar un algoritmo que permita encontrar la máxima similitud en regiones locales de las cadenas.

A primera instancia la solución del problema parece complicarse y elevar su complejidad. Para hacer el planteamiento del algoritmo consideremos nuevamente que lo que se intenta hacer es encontrar el alineamiento global óptimo entre todas las subcadenas de X y todas las subcadenas de Y. De inicio, encontrar todas las subcadenas de X toma $O(n_x^2)$ siendo n_X la longitud de X al igual que toma $O(n_Y^2)$ encontrar las subcadenas de Y siendo n_Y su longitud. Con lo anterior se parte de que tan solo la obtención de las subcadenas tarda $\Theta(n_X^2 n_Y^2)$. Si a ello se suma que el alineamiento global de cada par de subcadenas X^s y Y^s de longitudes n_X^s y n_Y^s respectivamente toma $O(n_X^s n_Y^s)$, se tiene que el cálculo del alineamiento local óptimo se lograría en $O(n_X^3 n_Y^3)$.

Afortunadamente, Temple Smith y Michael Waterman [12] han logrado una mejora significativa en la obtención del alineamiento local óptimo al hacerlo en $O(n_X n_Y)$ únicamente estableciendo una pequeña restricción. Esta restricción consiste en limitar el esquema de puntuación en las comparaciones de tal modo que se pueda asumir que el alineamiento global óptimo de dos cadenas vacias tiene valor 0.

Con esto, ya se puede empezar sugiriendo que los casos base son:

$$S_L(i,0) = 0$$

y

$$S_L(0,j) = 0$$

Por su parte, la recurrencia general es:

$$S_L(i,j) = Max(0, S_L(i-1,j-1) + t(i,j), S_L(i-1,j) + g, S_L(i,j-1) + g)$$

con $t(i,j) = M$ si $X[i] = Y[j]$ y $t(i,j) = m$ en otro caso. Las relaciones de recurrencia entonces obtenidas para el cálculo del alineamiento local óptimo se pueden describir de la forma:

$$S_L(i,j) = \begin{cases} 0 & i = 0 \ \vee \ j = 0 \\ Max \begin{cases} S(i-1,j) + g, \\ S(i,j-1) + g, \\ S(i-1,j-1) + M \\ 0 \end{cases} & i,j > 0 \ \wedge \ X[i] = Y[j] \\ Max \begin{cases} S(i-1,j) + g, \\ S(i,j-1) + g, \\ S(i-1,j-1) + m \\ 0 \end{cases} & i,j > 0 \ \wedge \ X[i] \neq Y[j] \end{cases} \tag{3.5}$$

Las recurrencias para el alineamiento local son casi idénticas a las del alineamiento global con la única diferencia de la inclusión de un cero en el caso del alineamiento

local. Esto puede resultar intuitivo dado que tanto para el alineamiento global como para el alineamiento local se van alineando dinámicamente los sufijos $X[1...i]$ y $Y[1...j]$, pero en el caso del alineamiento local, para el cálculo de ese sufijo puede que convenga ignorar el acumulado por tener una contribución negativa y comenzar nuevamente el cálculo desde cero lo cual no puede ocurrir en el caso del alineamiento global donde necesariamente toda la cadena es tomada en cuenta para la obtención de la medida de similitud. Dicho de otro modo, en la recurrencia implementada, el valor 0 puede actuar como el reinicio del cálculo de la medida de similitud local desde una posición más avanzada de cualquiera de las cadenas.

Una vez obtenida la matriz S_L, se tiene el valor del alineamiento local óptimo en la celda con valor máximo y para encontrar cuál es el alineamiento de las subcadenas que logra ese valor óptimo hay que hacer un recorrido en reversa a como se consiguió en la matriz S_L dicho valor. El recorrido inicia desde la celda de valor máximo y termina en la primera celda de valor 0 (sin considerar el alineamiento en esa celda).

El Algoritmo 17 se usa para llenar la matriz S_L y es también muy parecido al Algoritmo 16 usado para el llenado de S_G con las diferencias en los casos base y en la inclusión ya mencionada del valor 0 en la recurrencia general.

La Figura 3.7 muestra la matriz S_L calculada para la obtención de la medida de similitud de los dos alineamientos de la Figura 3.6. También se muestra el alineamiento local óptimo que existe entre las dos cadenas involucradas denotando entre ellas la operación de edición necesaria (Emparejamiento '=', Sustitución '*', Inserción '+', Eliminación '-').

En el análisis de la eficiencia del algoritmo de alineamiento local, se considera que dado que solo toma cuatro comparaciones por celda computar $S_L(i, j)$ y que el recorrido de la matriz sigue tomando $O(n_X n_Y)$, entonces el algoritmo de alineamiento local mantiene su orden $O(n_X n_Y)$.

La prueba a este resultado no se incluye en este trabajo pero es común encontrarse en la literatura [10].

Algoritmo 17 Cálculo de la medida de similitud local (alineamiento local)

```
LOCAL_SIMILARITY(X, Y)
   nx ← length(X)
   ny ← length(Y)
   for i ← 0 to nx do
      S[i, 0] ← 0
   for j ← 1 to ny do
      S[0, j] ← 0
   for i ← 1 to nx do
      for j ← 1 to ny do
         if X[i] = Y[i] then
            costCopying ← S[i-1, j-1] + M
         else
            costCopying ← S[i-1, j-1] + m
         costInserting ← S[i, j−1] + g
         costDeleting ← S[i−1, j] + g
         S[i, j] ← Max(0,costCopying,costInserting,costDeleting)
   return S[nx, ny]
```

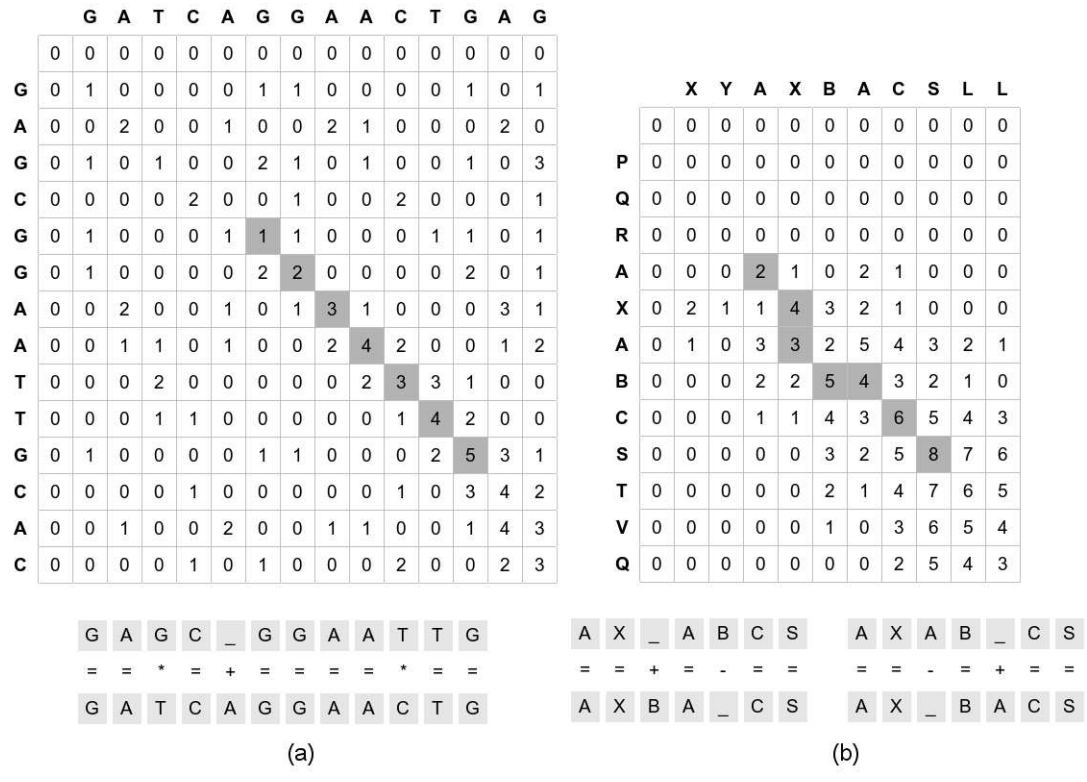


Figura 3.7: Matriz S_L resultado del algoritmo de similitud local con su respectivo alineamiento. (a) $M = 1$, $m = -1$ y $g = -2$ ($Similitud = 5$); (b) $M = 2$, $m = -2$ $g = -1$ ($Similitud = 8$)

$$\begin{array}{cccccccccccccccccc} C & A & G & C & A & \_ & C & T & T & G & G & A & T & T & C & T & C & G & G \\ \_ & \_ & \_ & C & A & G & C & G & T & G & G & \_ & \_ & \_ & \_ & \_ & \_ & \_ & \_ \end{array}$$

Figura 3.8: Alineamiento semiglobal óptimo entre dos cadenas con parámetros $M = 1$, $m = -1$ y $g = -2$ ($Similitud = 3$)

$$\begin{array}{cccccccccccccccccc} C & A & G & C & A & C & T & T & G & G & A & T & T & C & T & C & G & G \\ C & A & G & C & \_ & \_ & \_ & \_ & \_ & G & \_ & \_ & T & \_ & \_ & \_ & G & G \end{array}$$

Figura 3.9: Alineamiento global óptimo entre dos cadenas con parámetros $M = 1$, $m = -1$ y $g = -2$ ($Similitud = -12$)

3.3.3. Similitud semiglobal entre cadenas (alineamiento semiglobal)

En una comparación semiglobal, se intenta contabilizar el alineamiento de dos cadenas completas como se hace en el alineamiento global pero ignorando algunos de los espacios vacios que existen en las secuencias al inicio o al final de estas. Por ejemplo, en el alineamiento de la Figura 3.8, el espacio vacio que existe en la primera secuencia no sería ignorado, mientras que en la segunda cadena todos los espacios serían ignorados por el hecho de estar antes del inicio de la cadena o después del final de la misma [13].

Hay que notar que la longitud de estas dos cadenas varía considerablemente. Mientras que una tiene una longitud de 18, la otra tiene longitud de 8 lo cual contribuye negativamente en gran medida a la puntuación de lo que sería el alineamiento global que con los parámetros $M = 1$, $m = -1$ y $g = -2$ estaría dando -19.

El anterior definitivamente no es el mejor de los alineamientos globales entre esas dos secuencias puesto que el alineamiento de la Figura 3.9 lo supera alineando todos los elementos de la segunda cadena con elementos idénticos en la primera, aunque para ello tenga que fragmentar por completo toda la segunda cadena y obtener así una puntuación de -12.

Este segundo alineamiento, obtenido del concepto de alineamiento global óptimo, dependiendo de la aplicación, puede no ser de interés debido a que se olvida de la

idea de encontrar una región similar dentro de las secuencias con el fin de maximizar la función objetivo. Contrario a eso, este segundo alineamiento lo que hace es separar la secuencia con el único fin de obtener la mejor puntuación posible sin considerar que muchas veces resulta más atractivo un alineamiento como el de la Figura 3.8 aun teniendo una puntuación menos favorable en el alineamiento global.

Si por otro lado, se permite alinear las cadenas completas de forma parecida a la de un alineamiento global, pero sin penalizar por los espacios de alineamiento al inicio y al final, se favorece a el mejor alineamiento en un segmento compacto y con alta similitud como ocurre en el alineamiento local pero con la diferencia de que en este caso sí se estarían considerando ambas cadenas completas.

Para el caso del alineamiento de las cadenas de la Figura 3.8, la puntuación sin penalizar los espacios anteriores y posteriores de la segunda secuencia es 3, producto de 6 emparejamientos, por tan solo 1 remplazo y 1 eliminación. Esta medida bien supera la obtenida por el alineamiento de las secuencias en la Figura 3.9.

La implementación de este tipo de alineamiento puede ser logrado con ligeras modificaciones al algoritmo de alineamiento global (Algoritmo 16). El primer cambio consiste en modificar los casos base de modo que se inicialice en 0 la primera fila y columna de la matriz de similitud semiglobal, evitando así la penalización por los espacios al inicio de la primera y de la segunda cadena. Los casos base son entonces:

$$S_S(i, 0) = 0$$

y

$$S_S(0, j) = 0$$

La recurrencia general se mantiene:

$$S_S(i, j) = Max(S_S(i-1, j-1) + t(i, j), S_S(i-1, j) + g, S_S(i, j-1) + g)$$

Las relaciones de recurrencia obtenidas se pueden describir:

$$S_S(i,j) = \begin{cases} 0 & i = 0 \ \vee \ j = 0 \\ Max \begin{cases} S_S(i-1,j) + g, \\ S_S(i,j-1) + g, \\ S_S(i-1,j-1) + M \end{cases} & i,j > 0 \ \wedge \ X[i] = Y[j] \\ Max \begin{cases} S_S(i-1,j) + g, \\ S_S(i,j-1) + g, \\ S_S(i-1,j-1) + m \end{cases} & i,j > 0 \ \wedge \ X[i] \neq Y[j] \end{cases} \tag{3.6}$$

Finalmente, la despenalización de los espacios al final de las dos cadenas es algo que no muestra la fórmula 3.6 y que se logra hasta el momento de seleccionar la máxima puntuación del alineamiento. El alineamiento semiglobal óptimo lo da la celda de la fila n_X o de la columna n_Y en la matriz S_S y que tiene el valor máximo. El Algoritmo 18 describe el cálculo del alineamiento semiglobal.

La Figura 3.10 muestra la matriz S_S resultado del cálculo de la medida de similitud semiglobal entre las cadenas de la Figura 3.8. También se muestra el alineamiento semiglobal óptimo que existe entre las dos cadenas involucradas denotando entre ellas la operación de edición necesaria (Emparejamiento '=', Sustitución '*', Inserción '+', Eliminación '-', Ignorado '#').

El análisis de la eficiencia en tiempo y espacio del algoritmo de alineamiento semiglobal no cambia con respecto al de alineamiento global, lo cual nos mantiene un orden $O(n_X n_Y)$.

Algoritmo 18 Cálculo medida de similitud semiglobal (alineamiento semiglobal)

```
SEMIGLOBAL_SIMILARITY(X, Y)
   nx ← length(X)
   ny ← length(Y)
   for i ← 0 to nx
      S[i, 0] ← 0
   for j ← 1 to ny
      S[0, j] ← 0
   for i ← 1 to nx
      for j ← 1 to ny
         if X[i] = Y[i] then
            costCopying ← S[i-1, j-1] + M
         else
            costCopying ← S[i-1, j-1] + m
         costInserting ← S[i, j-1] + g
         costDeleting ← S[i-1, j] + g
         S[i, j] ← Max(costCopying,costInserting,costDeleting)
   return Max(S[nx, 0],...S[nx, ny], S[0, ny],...S[nx, ny])
```

		C	A	G	C	A	C	T	T	G	G	A	T	T	C	T	C	G	G
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	1	-1	-1	1	-1	1	-1	-1	-1	-1	-1	-1	-1	1	-1	1	-1	-1
A	0	-1	2	0	-1	2	0	0	-2	-2	-2	0	-2	-2	-1	0	-1	0	-2
G	0	-1	0	3	1	0	1	-1	-1	-1	-1	-2	-1	-3	-3	-2	-1	0	1
C	0	1	-1	1	4	2	1	0	-2	-2	-2	-2	-3	-2	-2	-4	-1	-2	-1
G	0	-1	0	0	2	3	1	0	-1	-1	-1	-3	-3	-4	-3	-3	-3	0	-1
T	0	-1	-2	-1	0	1	2	2	1	-1	-2	-2	-2	-2	-4	-2	-4	-2	-1
G	0	-1	-2	-1	-2	-1	0	1	1	2	0	-2	-3	-3	-3	-4	-3	-3	-1
G	0	-1	-2	-1	-2	-3	-2	-1	0	2	3	-1	-1	-3	-4	-4	-5	-2	-2

C	A	G	C	A	_	C	T	T	G	G	A	T	T	C	T	C	G	G
#	#	#	=	=	+	=	*	=	=	=	#	#	#	#	#	#	#	#
_	_	_	C	A	G	C	G	T	G	G	_	_	_	_	_	_	_	_

Figura 3.10: Matriz S_S resultado del algoritmo de similitud semiglobal con su respectivo alineamiento y usando los parámetros $M = 1$, $m = -1$ y $g = -2$ ($Similitud = 3$)

3.3.4. Consideraciones finales

3.3.4.1. Terminología dada para el alineamiento global y local

Muchos de los algoritmos tratados en este capítulo, son en sí temas completos que se enfocan a la solución de problemas de Biología. En la literatura especializada en Biología, comúnmente el alineamiento o similitud global es referido como el alineamiento o similitud de Needleman-Wunsch [11] dado que son las personas a las que se les atribuye la presentación del problema de similitud global. Más o menos por razones parecidas, el alineamiento o similitud local es referido como el alineamiento o similitud Smith-Waterman [12].

De cualquier modo, es conveniente precisar la confusión que existe en parte de la literatura, ya que de manera errónea atribuyen también la propuesta de solución de alineamiento global a Needleman-Wunsch cuando en realidad la propuesta de estos autores corría en tiempo cúbico y la que conocemos ahora es una mejora a esa propuesta original que corre en tiempo cuadrático.

Respecto al método de solución de Smith-Waterman que corre en tiempo cuadrático, sí puede atribuirse a los autores ya que ellos presentan tanto el problema como la solución actualmente usada para encontrar el alineamiento local. De cualquier modo, muchas de las soluciones para el alineamiento local también llegan a tener algunas diferencias con respecto a la propuesta original de Smith-Waterman.

3.3.4.2. Selección de los parámetros en la comparación de secuencias

Es importante entender que el éxito de los algoritmos generales de similitud entre cadenas en mucho depende de la correcta selección de los parámetros de comparación.

En dependecia de la aplicación para la que se diseña, varios detalles se deben tener presentes al momento de la selección del sistema de puntuación que será usado. Principalmente, esos detalles estarán afectando la puntuación de un emparejamiento M, una sustitución m y una eliminación g. Las sugerencias para establecer esos

parámetros de comparación serán dados con base a lo que los algoritmos de similitud requieren. Sin embargo, muchas de esas sugerencias también pueden ser adaptadas para el trabajo con los algoritmos de distancias de edición.

En cualquier sistema de puntuación siempre es importante que el valor de un emparejamiento sea mayor que el de una sustitución o eliminación para así alentar el alineamiento de elementos idénticos entre las cadenas. Otra sugerencia muy básica y que normalmente es usada es la de asignar un valor de eliminación $g < 0$ y de sustitución $m < 0$ dado que éstos deben influir de manera negativa en el resultado de la medida de similitud. Comúnmente se recomienda también que una sustitución sea preferida sobre el uso de un par de espacios en blanco. Por ejemplo, el siguiente alineamiento que aparece del lado izquierdo regularmente es decidido que tenga una puntuación mayor que el alineamiento del lado derecho:

$$\begin{array}{cc} A & \_ \; A \\ C & C \; \_ \end{array}$$

Las sugerencias anteriormente mencionadas nos llevan a definir una regla comúnmente aplicada y que se puede definir con la desigualdad

$$2g < m < M$$

Es de notar que si todos los factores de la desigualdad son multiplicados por una constante positiva, el alineamiento óptimo se mantine proporcionalmente igual. Esta propiedad puede ser útil cuando se desea transformar todos los valores del sistema de puntuación a valores enteros con el fin de lograr un procesamiento más rápido.

Considerar ahora las transposiciones que pueden ocurrir entre dos cadenas. Si se tienen las secuencias AT y TA, existen tres posibles alineamientos entre ellas de dos tipos esenciales:

$$\begin{array}{ccc} A \; T & A \; T \; \_ & \_ \; A \; T \\ T \; A & \_ \; A \; T & A \; T \; \_ \end{array}$$

El primer tipo de alineamiento es el de más a la izquierda cuya puntuación está dada por $2m$ y el segundo tipo se presenta en los alineamientos del centro y de la derecha cuya puntuación en cualquiera de los dos casos queda dada por $M + 2g$. Entonces, si m es igual a la media aritmética de M y $2g$, es decir $m = \frac{M+2g}{2}$, los dos tipos de alineamiento previo serán equivalentes en términos de su medida de similitud. Para dar preferencia a uno de ellos sobre otro, se debe recorrer m hacía cualquiera de los dos extremos: M o $2g$.

En la misma línea, es posible imaginar otros ejemplos de alineamiento entre un par de secuencias cortas y establecer cuál de los posibles alineamientos sería el más deseado. Por ejemplo, es posible que entre los alineamientos

$$\begin{array}{ccccc} A & T & C & G & \_ \\ \_ & T & C & G & A \end{array} \qquad\qquad \begin{array}{cccc} A & T & C & G \\ T & C & G & A \end{array}$$

se prefiera el alineamiento de la izquierda, lo cual se logra con la desigualdad $4m < 3M + 2g$.

Aunque al momento se ha trabajado con sistemas de puntuación arbitrarios, en la práctica se debe escoger muy bien el sistema a usar. Actualmente ya existen sistemas más sofisticados que una simple asignación arbitraria de M, m y g.

En el campo de la Biología un emparejamiento entre aminoácidos con propiedades físicas o químicas similares, como tamaño, carga o hidrofobicidad, suele recibir una mejor puntuación que el emparejamiento entre otros aminoácidos que no son tan parecidos. Más aun, en este contexto, los sistemas de puntuación generales ya no son suficientes y se tiene que recurrir a sistemas particularmente establecidos de acuerdo a la aplicación en la que se usan. Las matrices *PAM*, por ejemplo, son comunes para trabajar en la comparación de proteínas [13].

En el siguiente capítulo se hará uso de algoritmos anteriormente citados para ser aplicados en la comparación de códigos de cadena. En algunos momentos será necesario trabajar con parámetros de comparación estáticos como se ha hecho hasta el momento, pero también se analizará un sistema de puntuación más dinámico en el que habrá necesidad de sustituir $t(i, j)$, que hasta ahora había sido definida como

la función que regresa el valor de similitud de $X[1...i-1]$ y $Y[1...j-1]$ más M en caso de que $X[i] = Y[j]$ o más m en otro caso.

La función $t(i, j)$ será remplazada por la función s la cual recibe los valores $X[i]$ y $Y[j]$ de la cadenas a ser evaluados, de modo que el llamado $s(X[i], Y[j])$ devuelve el puntaje del alineamiento entre los elementos $X[i]$ y $Y[j]$. Nótese que entonces la función s ahora absorbe el uso de los parámetros de comparación M y m y si agregamos que el valor de una eliminación g también debe convertirse en un caso particular de la función s con un llamado del tipo $s(X[i],'\_')$ o $s('\_', Y[j])$, sabremos entonces que en la función de comparación s queda concentrado todo el sistema de puntuación. Internamente la función s se representa como una matriz de sustitución en la que ya están precálculados los valores de cada posible par de entradas. *PAM* y *BLOSUM* son dos ejemplos de matrices ampliamente usadas en aplicaciones biológicas [13].

3.3.4.3. Mejoras a los algoritmos

Los algoritmos presentados en este capítulo se definen en su forma más genérica, pero también es bueno señalar que existen muchas variantes con algunas pequeñas mejoras en espacio o en tiempo de ejecución o en otros casos con considerables mejoras tras haberle planteado algunas restricciones al problema. Se deja a apreciación del que diseña la solución a un problema afín la decición de escoger que mejoras son las que conviene aplicar.

Solo por mencionar un ejemplo, para la reducción de la complejidad de cualquiera de los algoritmos de alineamiento se puede acotar el número de errores máximos permitidos entre dos cadenas para que cuando este máximo se rebase, el cálculo de la matriz de similitud se suspenda considerando que las cadenas ya no son lo suficientemente parecidas como para ser de interés. Como esta restricción existen otras que según el interés del que las implementa pueden ser aplicadas con mejoras considerables en el rendimiento del algoritmo.

3.3.4.4. Implementaciones en bases de datos

Con el camino del desarrollo de aplicaciones cada vez más completas y con un nivel de complejidad cada vez más alto en ellas, la necesidad de integrar mejoras en distintos ámbitos se vuelve crucial. Este ha sido el caso de la integración de grandes bases de datos para el almacenamiento de secuencias de gran longitud con alguna representación biológica, donde la recuperación de datos después de haberlas comparado es muy usada. Esto también trajó consigo una necesidad de mecánismos eficientes en el procesamiento de esos grandes cúmulos de datos respondiendo a cualquier consulta lanzada a la base de datos. Por ejemplo, en una aplicación típica, uno puede tener una secuencia que funcionará como consulta para encontrar todas aquellas secuencias en la base de datos que se le parezcan, lo cual significa quizá cientos de miles de comparaciones entre secuencias.

La complejidad cuadrática de los algoritmos que se han tratado hasta el momento, puede entonces no ser lo suficientemente eficiente para trabajar con bases de datos demasiado grandes. Con el fin de hacer más veloces las búsquedas, han sido desarrollados métodos novedosos y muy rápidos. En general, estos métodos se basan en cuestiones heurísticas y en muchas consideraciones relacionadas a la aplicación para la que fueron diseñados, y por tanto, es díficil establecer teóricamente su complejidad en tiempo y espacio pero se puede decir que en ocasiones llegan a representar mejoras significativas al trabajar sobre grandes volúmenes de datos.

Los programas que permiten este tipo de búsquedas en grandes bases de datos regularmente no usan Programación Dinámica, o si acaso corren algunas variantes de ella. Siguiendo con las líneas de aplicaciones biológicas, tal vez los programas más importantes de este tipo son *FASTA* y *BLAST* [13].

Como ya se ha mencionado, no es interés de este trabajo adaptarse a alguna solución solo por ser conveniente para una aplicación particular. Por el contrario, se intenta presentar soluciones desde un enfoque general y cuando existe oportunidad se sugiere alguna mejora particular quedando a juicio del que la considera implementar.

Capítulo 4

Propuesta de un algoritmo de semejanza entre formas discretas

Para entregar una medida que permita saber que tan parecidas son dos formas discretas hay muchos trabajos al respecto [4, 5] con resultados éxitosos algunos de ellos. Uno de los principales retos en este trabajo es presentar otro método que permita igualmente la obtención de una medida de similitud entre formas discretas pero a partir de principios distintos.

Considerando la bondad que brindan los códigos cadena para representar fielmente una forma discreta, significando también un ahorro de espacio de almacenamiento y de tiempo de procesamiento, parece atractiva la posibilidad de obtener esa medida de similitud a partir de la representación de formas con un código de cadena.

Más aun, conociendo los trabajos existentes en otras áreas de la ciencia para la obtención de alineamientos óptimos entre secuencias, principalmente en áreas de la Biología, luce interesante tratar de combinar ambas aproximaciones para dar solución a un problema que hasta el momento había sido abordado desde una perspectiva muy distinta. En este caso el cálculo de una medida de similitud entre formas discretas sin aludir directamente a la representación de la imagen en el plano cartesiano.

Es entonces cuando nace la idea en la que se basa este trabajo que intenta

representar una forma discreta con un código de cadena que después podrá ser comparado con el código de cadena que representa a otra forma discreta y todo esto usando algoritmos de alineamiento clásicos.

El primer paso entonces, es la selección de un código de cadena lo suficientemente robusto y que se ajuste a la naturaleza del problema que se quiere resolver. Inicialmente se considero usar el Código Cadena Vértice (*Vertex Chain Code, VCC*) [1] que entre sus muchas propiedades se encuentra su invarianza a la rotación. Sin embargo, en la misma ventaja de ese código está la dificultad para usarlo en el alineamiento entre cadenas ya que, en *VCC* un valor de la cadena representa la cantidad de celdas del objeto en contacto con un vértice, lo cual define el camino a seguir a partir de cada vértice del contorno, pero sin dar información de la orientación que hay en el segmento de dicho vértice. A partir de ello se garantiza la invarianza a la rotación, pero esto también hace muy complicado el poder usarlo para comparar segmentos del contorno de una forma. Como ejemplo, hace falta solo obtener la representación *VCC* de cualquier curva discreta y compararla consigo misma habiendo modificado solo el valor de un elemento a la mitad de la cadena. Aunque las cadenas que se comparan aun siguen siendo altamente similares, la curva que representa puede cambiar por completo. Algo similar ocurre con *SACC'* (primera derivada *SACC*).

Es decir, para una forma apreciada a simple vista, la información de la orientación que tiene cada segmento del contorno es de gran importancia y a partir de la comparación de dicho segmento con el de otra forma, es posible decidir si se trata o no de segmentos iguales basándose en la orientación que estos tienen. A mayor cantidad de segmentos iguales dado cierto orden entre dos cadenas, mayor será la medida de similitud entre las dos formas representadas por esos códigos.

Posteriormente, se propusó otro código cadena, también presentado en este trabajo, denominado Código Cadena de Pendiente Acumulada (*Slope Accumulation Chain Code, SACC*) que es un código que indica la cantidad de incrementos en la pendiente que se hacen para llegar a cada uno de los vértices sobre el contorno. Este código no es invariante al punto de inicio, e indirectamente tampoco lo es a la

rotación y en cada elemento va implícita la orientación del segmento que representa. En ese sentido es parecido al código cadena de Freeman (*Freeman Chain Code, FCC*) [3]. Con una variante de este nuevo código, es posible usar los algoritmos de similitud entre cadenas con muy buenos resultados mismos que serán presentandos en este capítulo.

Es importante notar que el cálculo de esta medida trabaja directamente sobre el mundo unidimensional de cadenas y no sobre imágenes representadas en el plano cartesiano.

En adelante se hará uso de la base de datos de imágenes de especies marinas[1] [14] para ilustrar el funcionamiento de cada uno de los algoritmos usados. En el Apéndice A se puede encontrar la base de datos completa, y en el transcurso del capítulo se hará referencia a las imágenes que en ésta hay.

4.1. Normalización de cadenas

Para la obtención de una medida de similitud entre formas discretas representadas por un código de cadena, primero se debe de obtener dicha representación, en este caso con el código *SACC*. La implementación de un algoritmo de este tipo es en realidad muy simple y solo debe usarse una vez ya que en lo consiguiente se trabaja directamente con la cadena obtenida.

Sin embargo, existe un primer obstáculo que es la resolución a la que están representadas las imágenes. A mayor resolución de una imágen binaria, mayor será la longitud del código cadena que de ésta se obtenga.

Esto cobra importancia cuando dos cadenas sean comparadas para el cálculo de su similitud. Que dos cadenas difieran notablemente en su longitud, puede afectar significativamente en detrimento de la medida de similitud entre ellas. Esto ocurre incluso cuando se trate de la misma forma discreta representada a distintas escalas,

[1] Proporcionada por Sadegh Abbasi (`http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/a/Abbasi:Sadegh.html`)

ya que en ese caso su medida de similitud no será tan buena como se cosideraría que debe ser.

Una idea para mitigar en parte este problema, consiste en escalar cada representación de una forma discreta lo suficente para que todas las cadenas que sean comparadas tengan una longitud parecida. Para ello, se puede hacer uso de los algoritmos de escalamiento presentados anteriormente y concentrar así el problema en la elección de los factores de escala r_x y r_y.

Dado que se quiere que las proporciones de la imagen se mantengan, $r_x = r_y$ es una condición que se debe cumplir. Finalmente, hay que saber también cuál es el factor de normalización F. Es decir, F es la magnitud a la cual tratamos de aproximar la forma discreta ya sea en área o en perímetro.

4.1.1. Normalización en área

Una idea para normalizar una forma discreta a ser comparada es en área. Para ello hay que calcular cuál es el área actual A de la forma discreta que concretamente es equivalente a contar el número de píxeles del objeto. Una vez conocida A, se pude aplicar el escalamiento a la cadena $SACC$ con los factores

$$r_x = r_y = \frac{F}{A} \tag{4.1}$$

4.1.2. Normalización en perímetro

La otra forma para normalizar y que estará siendo usada en este capítulo, consiste en trabajar directamente sobre el número de elementos n de una cadena $SACC$. Para normalizar la cantidad de elementos de la cadena en un factor de normalización F, basta con escalar la representación de la forma discreta directo desde la cadena con los factores

$$r_x = r_y = \frac{F}{n} \tag{4.2}$$

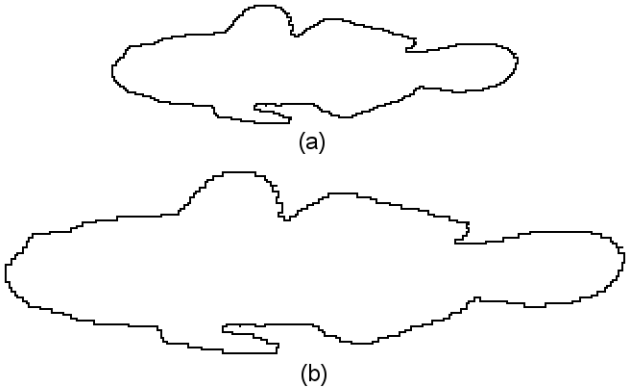


Figura 4.1: Ejemplo de normalización en perímetro $F = 1000$. (a) *img0930* con $n = 656$; (b) *img0930* con $n = 998$ logrado con el factor de escalamiento $r = 1{,}524390244$

Para muchas aplicaciones es posible que no se quiera pasar por un proceso de normalización de las cadenas considerando importante no solo el análisis de la forma, sino el tamaño que éstas tengan. Para el caso de este trabajo, se manejan las figuras presentadas en el Apéndice A. Esas imágenes se sabe que son imágenes binarias y que el objeto que en ellas aparece ha sido previamente segmentado, aislando así otros problemas cuya solución no son la finalidad de este trabajo.

La base de datos presenta 1100 formas discretas representando distintas especies marinas. Una vez obtenida esa base de datos, las imágenes pasaron por un proceso de binarización y extracción del código *SACC*, el cual fue normalizado en perímetro con un factor de normalización $F = 500$ escalando las formas discretas lo necesario. A pesar de esto, la cantidad de elementos de las cadenas puede seguir variando entre sí, pero esto ocurre en un margen mucho más estrecho, en el cual, la cantidad de elementos se acerca bastante a 500 elementos.

La Figura 4.1(a) muestra un ejemplo en el que *img0930* con un código *SACC* de $n = 656$ cantidad de elementos fue normalizado en perímetro con el factor $F = 1000$ dando por resultado un factor de escalamiento $r_x = r_y = 1{,}524390244$ con el que se obtuvo la Figura 4.1(b).

4.2. Medida de similitud global entre formas discretas representadas por su código SACC

Como se mencionó en el capítulo anterior, el problema de encontrar la medida de similitud entre dos cadenas es equivalente al de encontrar sus alineamientos óptimos. Para el caso de la medida de similitud global, el alineamiento óptimo que se busca se debe dar entre ambas cadenas completas.

El algoritmo de similitud global ya comentado (Algoritmo 16), puede ser aplicado para encontrar la medida de similitud. En ese caso, es importante considerar que en el algoritmo de similitud global ambas cadenas son comparadas por completo.

Es también de apreciar que dependiendo de la aplicación, es posible que se quiera obtener la similitud entre dos cadenas que representan una forma discreta en cualquiera de su 4 posibles orientaciones sobre celdas regulares rectangulares. Para ello únicamente hay que tomar la máxima similitud entre la primera cadena y la segunda sin rotar, la primera y la segunda rotada 90°, la primera y la segunda rotada 180° y finalmente la primera y la segunda rotada 270°.

Otra posible transformación de interés es la del reflejo de una imagen. Para ambos casos, los algoritmos sobre cadenas *SACC* ya han sido propuestos en el capítulo en el que se presentó *SACC* (Algoritmo 1 y Algoritmo 2).

También es importante considerar lo suceptible que se vuelve la medida de similitud al punto de inicio aun considerando rotaciones. Por ejemplo, para el caso de *img0392* y de *img0400* que son muy parecidas entre sí, la medida de similitud estará por debajo de lo esperado en la escala de clasificación de similitud. Esto ocurre porque a pesar de ser formas muy parecidas entre sí, el cálculo de las cadenas *SACC* que representan estas formas se hace desde posiciones de inicio distintas lo cual termina dando dos cadenas también muy distintas y en cuyo caso carece de sentido encontrar el valor de similitud entre esas cadenas tan distintas.

Hasta este momento, se ha usado como punto de inicio de una cadena *SACC* el primer punto notable que, como se había indicado anteriormente, es el primer vértice

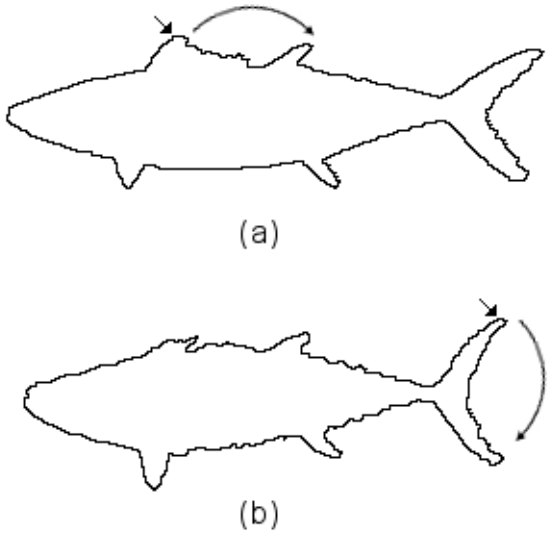


Figura 4.2: Ejemplo de problemas de escoger el primer punto notable como punto de inicio (a) *img0392;* (b) *img0400*

del contorno del objeto que se encuentra al barrer la imagen de izquierda a derecha y de arriba a abajo (Cuadro 2.2). La Figura 4.2 muestra las imágenes *img0392* y de *img0400* destacando sus respectivos puntos de inicio.

Una primera aproximación para resolver este problema es usar la medida de similitud local entre cadenas para contabilizar únicamente los segmentos de alta similitud e ignorar el resto de la cadena que penaliza el alineamiento.

4.3. Medida de similitud local entre formas discretas representadas por su código SACC

Como ya se ha visto, en algunos casos dos formas discretas representadas por su código *SACC* pueden ser a simple vista muy parecidas entre sí, y sin embargo, las cadenas que los representan pueden ser muy distintas debido a que el punto de inicio de una es diferente al punto de inicio de la otra. Este problema hace que el uso de la medida de similitud global entre formas no sea lo suficientemente buena

aun cuando se prueben todas las posibles rotaciones o transformaciones en espejo para alguna de las formas discretas.

Existe una posibilidad que consiste en probar la distancia de similitud global entre formas haciendo la comparación de la primera cadena contra todas las posibles cadenas resultado de un desplazamiento circular de una unidad en la segunda cadena. Esta solución es muy radical y extremadamente cara en el contexto del procesamiento que se tendría que hacer y más aun si para cada una de esas cadenas resultado del desplazamiento se prueban sus ocho posibles rotaciones-reflejos.

Otra solución interesante está en usar la similitud local para calcular la similitud entre las cadenas únicamente con el segmento de alta similitud entre ellas e ignorando aquellos segmentos que no tienen tanta similitud y penalizan la medida de similitud obtenida. La Figura 4.3 muestra nuevamente a *img0392* y a *img0400* marcando sus puntos de inicio y el segmento de alta similitud que seguramente sería considerado para el cálculo de la medida de similitud local. Es importante notar que en caso de usar el algoritmo de similitud global, el segmento inicial de la primera cadena y el segmento final de la segunda cadena penalizarían la similitud entre las dos imágenes aun cuando éstas son altamente similares.

Además del uso que se ha dado a la medida de similitud local existe otra aplicación para la que se puede usar en la comparación de cadenas *SACC*. En esta aplicación se considera que las cadenas no están normalizadas y se está buscando alguna región en común entre las dos cadenas. El algoritmo de similitud local en ese caso permitiría encontrar esa región de alta similitud como ocurriría entre las cadenas que representan las formas discretas de la Figura 4.4.

La Figura 4.5 muestra los resultados de buscar un patrón dentro de todas las imágenes de la base de datos usando la similitud local. El patrón usado se muestra en la Figura 4.5(a) y fue extraído de la *img0392* lo cual causa sentido al ver que la lista de las imágenes de mayor similitud local es encabezada por la misma *img0392*.

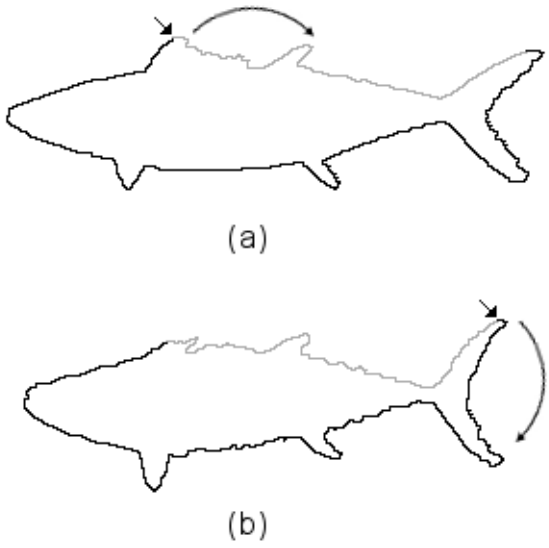


Figura 4.3: Mejora al problema de escoger el primer punto notable como punto de inicio usando cálculo de similitud local. (a) *img0392*; (b) *img0400*. Para ambos contornos la parte resaltada es la que encontraría el algoritmo con una región de alta similitud

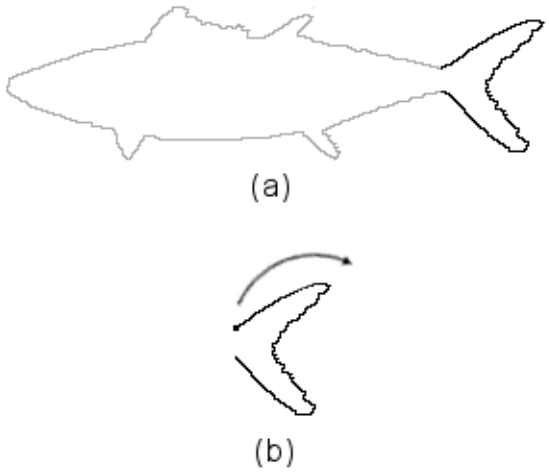


Figura 4.4: Otra aplicación de la medida de similitud local para encontrar regiones de alta similitud en imágenes representadas por su código $SACC$. (a) *img0392*; (b) Segmento a encontrar en el contorno de *img0392*

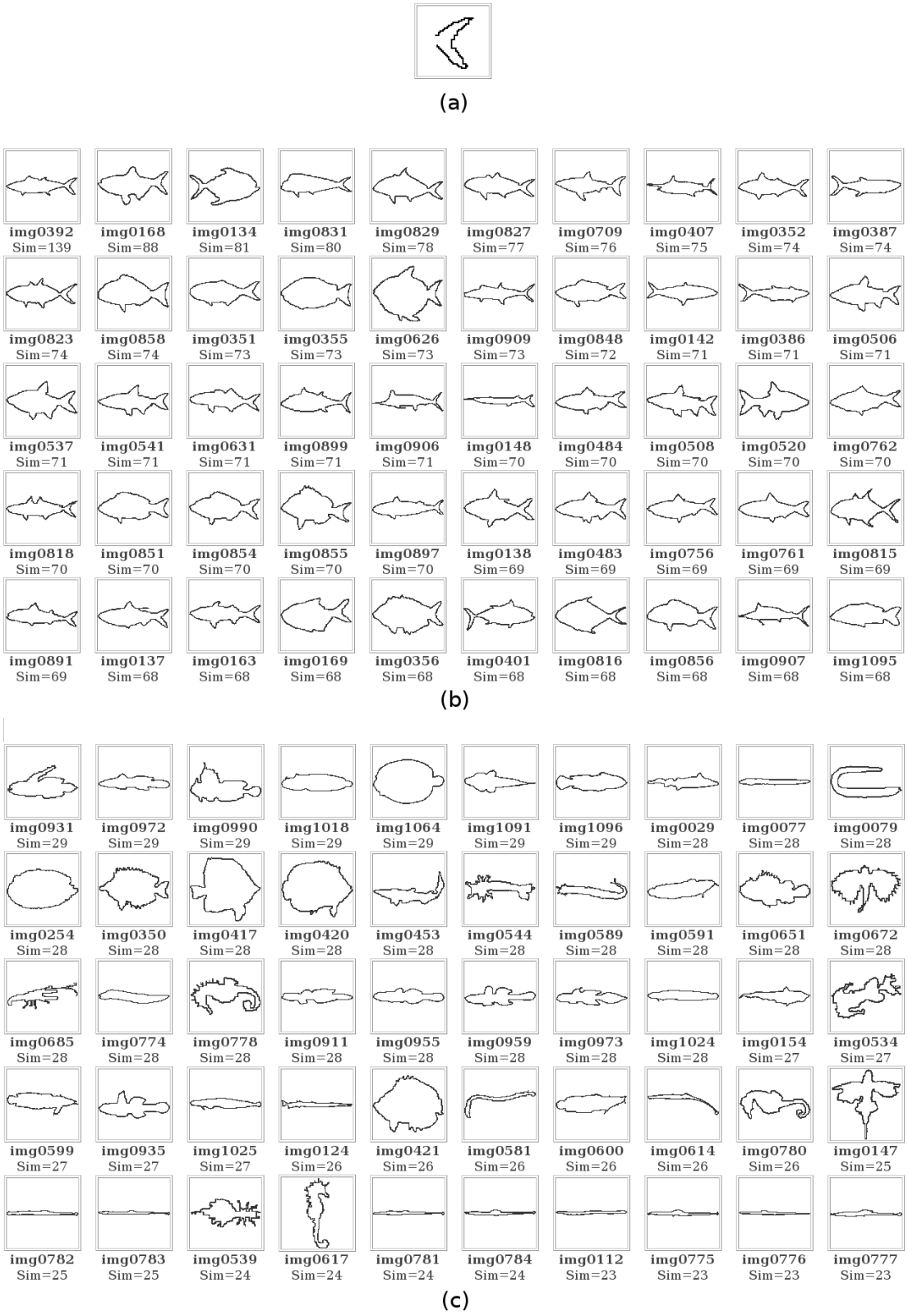


Figura 4.5: Resultado de encontrar la región de alta similitud en imágenes representadas por su código *SACC*. (a) patrón de búsqueda (extraído de *img0392*); (b) imágenes de mayor similitud local al patrón de búsqueda; (c) imágenes de menor similitud local al patrón de búsqueda

También es interesante notar que en esa lista de imágenes de alta similitud local, el patrón de búsqueda se presenta de una u otra manera en cada imagen, aun cuando se trate de peces muy distintos, Figura 4.5(b). Finalmente, en la Figura 4.5(c) se muestran las imágenes de menor similitud local al patrón de búsqueda utilizado y por ello se puede apreciar la ausencia del patrón de búsqueda en dichas imágenes.

Sin embargo, volviendo al caso de la comparación entre *img0392* y *img0400* aun queda un problema por resolver. Si bien es cierto que ahora la medida de similitud se está calculando únicamente sobre un segmento de alta similitud, sigue habiendo otro segmento que debería ser importante para describir la similitud entre las formas discretas y que no está siendo tomado en cuenta. Es entonces cuando se llega a la necesidad de combinar las ventajas entre similitud local para resolver el problema del punto de inicio, y similitud global para poder comparar las cadenas completas.

4.4. Medida de similitud semiglobal entre formas discretas representadas por su código SACC

Con la comparación semiglobal, se considera completo el traslape de las dos cadenas para la obtención de la medida de similitud y si hubiera espacios en blanco al inicio o al final de las cadenas no penalizan la medida de similitud obtenida. Sin embargo, al trabajar con cadenas normalizadas con prácticamente la misma cantidad de elementos esa no es una gran ventaja. Además nuevamente se cae en el problema asociado al punto de inicio de la cadena.

Ahora bien, con una pequeña adaptación a la obtención de la medida de similitud semiglobal se pueden resolver ambos problemas de una manera muy adecuada y otorgando una medida de similitud entre cadenas que resuelve prácticamente cualquier problema que se presente en los dos tipos de similitud anteriores.

Esta adaptación consiste en duplicar la primera de las dos cadenas, de modo que la segunda cadena busque en que posición de la primera cadena duplicada logra una

mejor alineación global, pero sin penalizar por los espacios en blanco antes del inicio o después del final del alineamiento. Esto visto de otro modo, es una manera más inteligente y eficiente de lograr lo que se lograría con aquella primera aproximación de hacer un desplazamiento circular en los elementos de la segunda cadena para posteriormente calcular la similitud global. La Figura 4.6(a) muestra dos cadenas $SACC$ en el nivel 1 de anidamiento que representan a la misma forma discreta desde distintos puntos de inicio por lo que su alineamiento global no sería muy bueno. La Figura 4.6(b) muestra un alineamiento entre la primera de esas cadenas duplicada y la segunda cadena. En este segundo caso, si se usa la medida de similitud semiglobal ignorando así los espacios vacios, se puede apreciar que el resultado será que ambas cadenas representan a la misma forma discreta lo cual es cierto.

Finalmente, para mejores resultados hay que obligar a que la segunda cadena se compare completa contra la repetición de la primera cadena. Para ello hay que hacer otra pequeña adaptación al algoritmo de similitud semiglobal para que en la matriz de similitud S, la fila $i = 0$ se inicialice con $S(0, j) = j * g$, penalizando así el que no se tome completa la segunda cadena. Igualmente, al momento de elegir el valor máximo se hará solamente sobre la columna $j = n$.

La Figura 4.7 muestra un ejemplo de la matriz de similitud S para las cadenas de la Figura 4.6(b). Para ese caso la medida de similitud entre las dos cadenas tiene valor 16 usando los parámetros de comparación $M = 1$, $m = -1$ y $g = -2$. Eso significa que hubo 16 emparejamientos con valor 1 que equivalen a toda la cadena.

La Figura 4.8(b) es un ejemplo en el que se muestran las 50 formas discretas de la base de datos de especies marinas que son más similares a *img0392*, mostrada en la Figura 4.8(a), mientras que en la Figura 4.8(c) se muestran las más distintas a *img0392*, en ambos casos ordenadas de la más similar a la menos similar. Es importante notar que, con toda lógica, la misma imagen *img0392* es la de mayor similitud. Adicionalmente, en la Figura 4.9 se muestran otros 7 ejemplos con las 10 imágenes más similares de la base de datos para las 7 entradas dadas. Los resultados completos pueden ser vistos en la referencia [15].

1211222334143344 2223341433441211

(a)

1211222334143344121122233414334
2223341433441211

(b)

Figura 4.6: (a) Cadenas *SACC* en su nivel de anidamiento 1 que representan la misma forma desde distintos puntos de inicio; (b) Alineamiento entre la primera cadena duplicada y la segunda cadena

		2	**2**	**2**	**3**	**3**	**4**	**1**	**4**	**3**	**3**	**4**	**4**	**1**	**2**	**1**	**1**
	0	-2	-4	-6	-8	-10	-12	-14	-16	-18	-20	-22	-24	-26	-28	-30	-32
1	0	-1	-3	-5	-7	-9	-11	-11	-13	-15	-17	-19	-21	-23	-25	-27	-29
2	0	1	0	-2	-4	-6	-8	-10	-12	-14	-16	-18	-20	-22	-22	-24	-26
1	0	-1	0	-1	-3	-5	-7	-7	-9	-11	-13	-15	-17	-19	-21	-21	-23
1	0	-1	-2	-1	-2	-4	-6	-6	-8	-10	-12	-14	-16	-16	-18	-20	-20
2	0	1	0	-1	-2	-3	-5	-7	-7	-9	-11	-13	-15	-17	-15	-17	-19
2	0	1	2	1	-1	-3	-4	-6	-8	-8	-10	-12	-14	-16	-16	-16	-18
2	0	1	2	3	1	-1	-3	-5	-7	-9	-9	-11	-13	-15	-15	-17	-17
3	0	-1	0	1	4	2	0	-2	-4	-6	-8	-10	-12	-14	-16	-16	-18
3	0	-1	-2	-1	2	5	3	1	-1	-3	-5	-7	-9	-11	-13	-15	-17
4	0	-1	-2	-3	0	3	6	4	2	0	-2	-4	-6	-8	-10	-12	-14
1	0	-1	-2	-3	-2	1	4	7	5	3	1	-1	-3	-5	-7	-9	-11
4	0	-1	-2	-3	-4	-1	2	5	8	6	4	2	0	-2	-4	-6	-8
3	0	-1	-2	-3	-2	-3	0	3	6	9	7	5	3	1	-1	-3	-5
3	0	-1	-2	-3	-2	-1	-2	1	4	7	10	8	6	4	2	0	-2
4	0	-1	-2	-3	-4	-3	0	-1	2	5	8	11	9	7	5	3	1
4	0	-1	-2	-3	-4	-5	-2	-1	0	3	6	9	12	10	8	6	4
1	0	-1	-2	-3	-4	-5	-4	-1	-2	1	4	7	10	13	11	9	7
2	0	1	0	-1	-3	-5	-6	-3	-2	-1	2	5	8	11	14	12	10
1	0	-1	0	-1	-2	-4	-6	-5	-4	-3	0	3	6	9	12	15	13
1	0	-1	-2	-1	-2	-3	-5	-5	-6	-5	-2	1	4	7	10	13	16
2	0	1	0	-1	-2	-3	-4	-6	-6	-7	-4	-1	2	5	8	11	14
2	0	1	2	1	-1	-3	-4	-5	-7	-7	-6	-3	0	3	6	9	12
2	0	1	2	3	1	-1	-3	-5	-6	-8	-8	-5	-2	1	4	7	10
3	0	-1	0	1	4	2	0	-2	-4	-5	-7	-7	-4	-1	2	5	8
3	0	-1	-2	-1	2	5	3	1	-1	-3	-4	-6	-6	-3	0	3	6
4	0	-1	-2	-3	0	3	6	4	2	0	-2	-3	-5	-5	-2	1	4
1	0	-1	-2	-3	-2	1	4	7	5	3	1	-1	-3	-4	-4	-1	2
4	0	-1	-2	-3	-4	-1	2	5	8	6	4	2	0	-2	-4	-3	0
3	0	-1	-2	-3	-2	-3	0	3	6	9	7	5	3	1	-1	-3	-2
3	0	-1	-2	-3	-2	-1	-2	1	4	7	10	8	6	4	2	0	-2
4	0	-1	-2	-3	-4	-3	0	-1	2	5	8	11	9	7	5	3	1
4	0	-1	-2	-3	-4	-5	-2	-1	0	3	6	9	12	10	8	6	4

Figura 4.7: Matriz de similitud S para las cadenas de la Figura 4.6(b) con los parámetros $M = 1$, $m = -1$ y $g = -2$ ($Similitud = 16$)

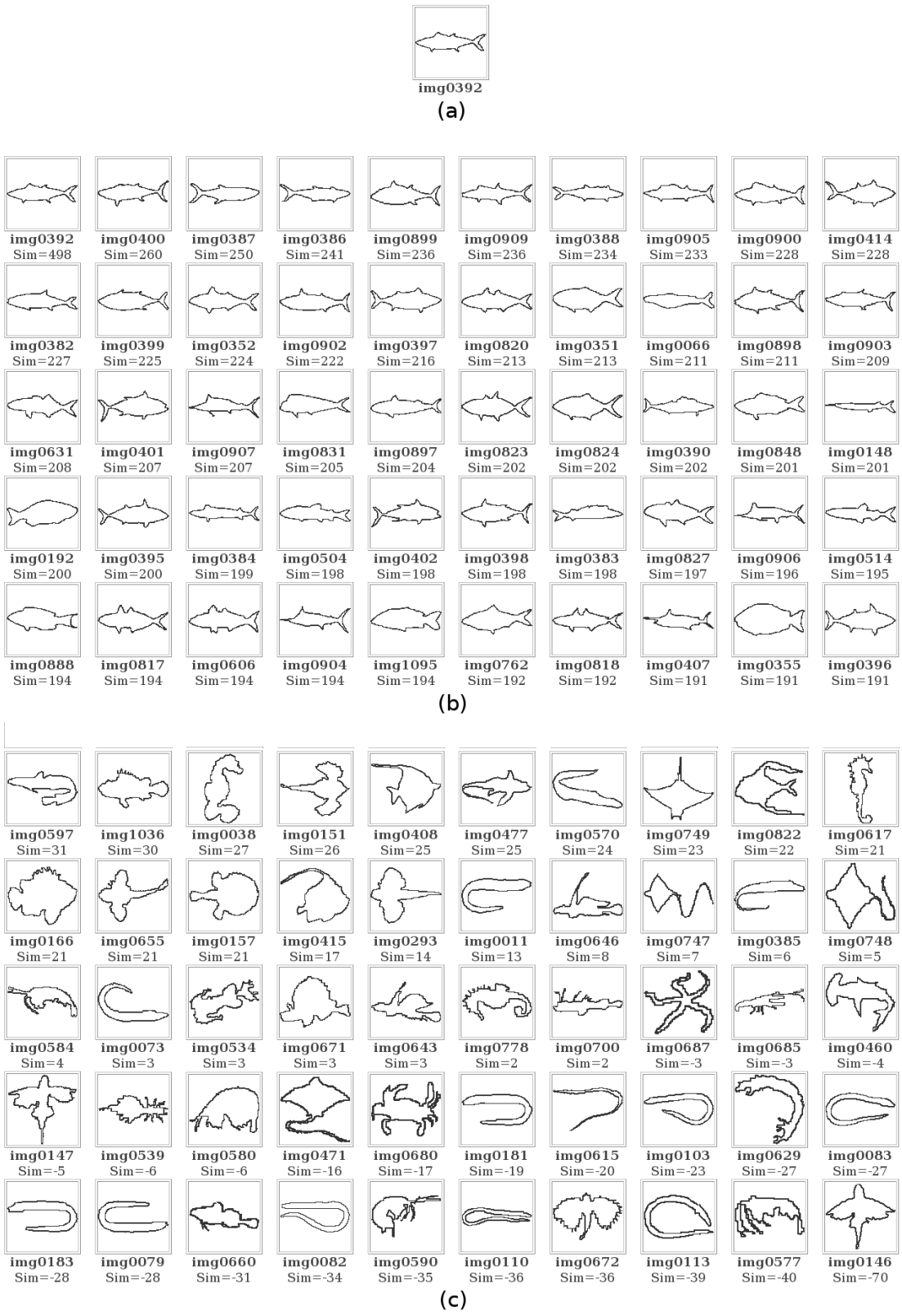


Figura 4.8: Similitud semiglobal con parámetros $M = 1$, $m = -1$ y $g = -2$. (a) *img0392* a comparar; (b) las 50 formas de mayor similitud; (c) las 50 formas de menor similitud

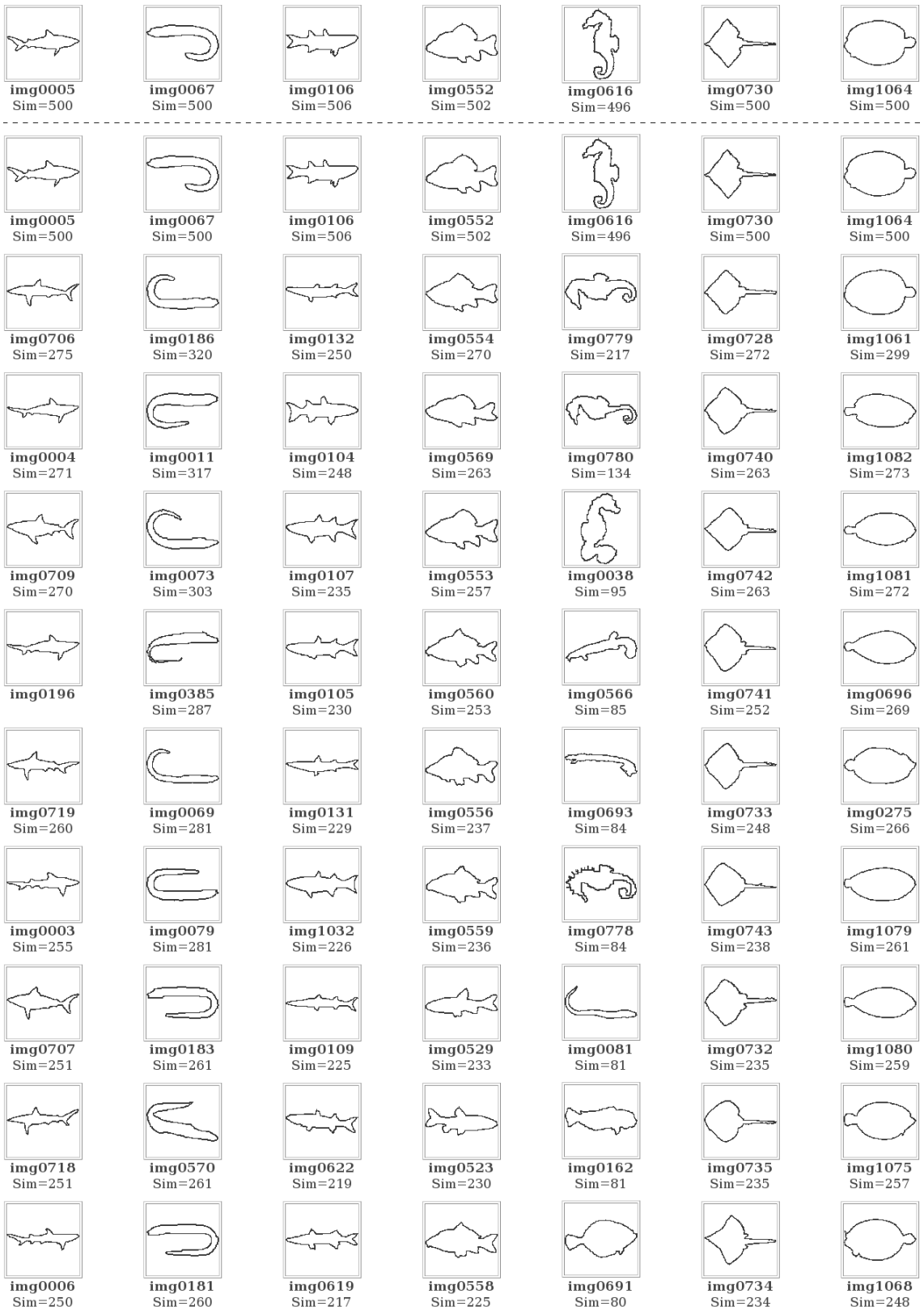


Figura 4.9: Similitud semiglobal con parámetros $M = 1$, $m = -1$ y $g = -2$. Sobre la línea punteada se encuentra la entrada dada y debajo las 10 imágenes de mayor similitud (ordenadas de mayor a menor)

4.5. Selección de los parámetros de comparación

Por ahora se ha probado con buenos resultados la medida de similitud semiglobal con unos parámetros de comparación $M = 1$, $m = -1$ y $g = -2$ que son usados para algunas aplicaciones de cálculo de similitud entre cadenas y en la literatura [10], además de cumplir con las características citadas en la sección 3.3.4.2 para ser considerado un sistema de comparación consistente. Sin embargo, dependiendo del contexto en el que trabajen esas cadenas, siempre es posible encontrar unos parámetros que se ajusten de mejor forma a la solución del problema.

La buena elección de estos parámetros, junto con algunas consideraciones importantes que se mencionan al final del capítulo, definen la precisión de la medida de similitud que se está calculando. En este caso, se considera entonces que los parámetros de comparación deben ser parte de los parámetros de calibración de cualquier sistema que intenta encontrar la similitud entre formas discretas mediante esta medida.

Para el uso de la medida de similitud semiglobal sobre cadenas $SACC$ en su nivel de anidamiento 1 donde todos los valores de la cadena están entre 1 y 4, una matriz de comparación que también fue usada con buenos resultados en este trabajo y que igualmente cumple con lo dicho en la sección 3.3.4.2, es la mostrada en el Cuadro 4.1. En esta matriz se promueve el emparejamiento de elementos sumando 4 a la medida de similitud por cada emparejamiento existente. Por el contrario, se penaliza con -3 a cada inserción o eliminación en una de las cadenas, con -2 por remplazos equivalentes a un incremento y finalmente con -4 a los remplazos de elementos con la dirección totalmente opuesta a su remplazo.

A este punto es importante entender la dualidad en la representación de los parámetros de comparación como se había hecho hasta el momento (usando los costos M, m y g) y como se hace ahora usando una matriz de comparación. En ocasiones resultará más cómodo referirse a la matriz ya que de otro modo se tienen que especificar las posibles condiciones que definirían cierto costo. Por ejemplo, para

	_	**1**	**2**	**3**	**4**
_	*	-3	-3	-3	-3
1	-3	4	-2	-4	-2
2	-3	-2	4	-2	-4
3	-3	-4	-2	4	-2
4	-3	-2	-4	-2	4

Cuadro 4.1: Tabla de comparación propuesta para cadenas *SACC* en su nivel de anidamiento 1

	_	**1**	**2**	**3**	**4**
_	*	-2	-2	-2	-2
1	-2	1	-1	-1	-1
2	-2	-1	1	-1	-1
3	-2	-1	-1	1	-1
4	-2	-1	-1	-1	1

Cuadro 4.2: Tabla de comparación para cadenas *SACC* en su nivel de anidamiento 1 equivalente a usar los parámetros de comparación $M = 1$, $m = -1$ y $g = -2$

la matriz del Cuadro 4.1, los parámetros de comparación son siempre $M = 4$ y $g = -3$, pero para el caso del remplazo son $m = -2$ cuando la diferencia entre los elementos comparados es igual a 1 y $m = -4$ en otro caso.

Igualmente, en el otro sentido, los parámetros de comparación que han sido usados hasta el momento: $M = 1$, $m = -1$ y $g = -2$, también pueden ser representados como una matriz (Cuadro 4.2).

La Figura 4.10 nuevamente muestran las 50 formas discretas más similares y las 50 formas discretas menos similares en la base de datos de especies marinas a *img0392*, pero ahora usando los parámetros de comparación del Cuadro 4.1. En la Figura 4.11 nuevamente se muestran 7 ejemplos con las las 10 imágenes más parecidas de la base de datos para las 7 entradas dadas. Los resultados completos pueden ser vistos en la referencia [15].

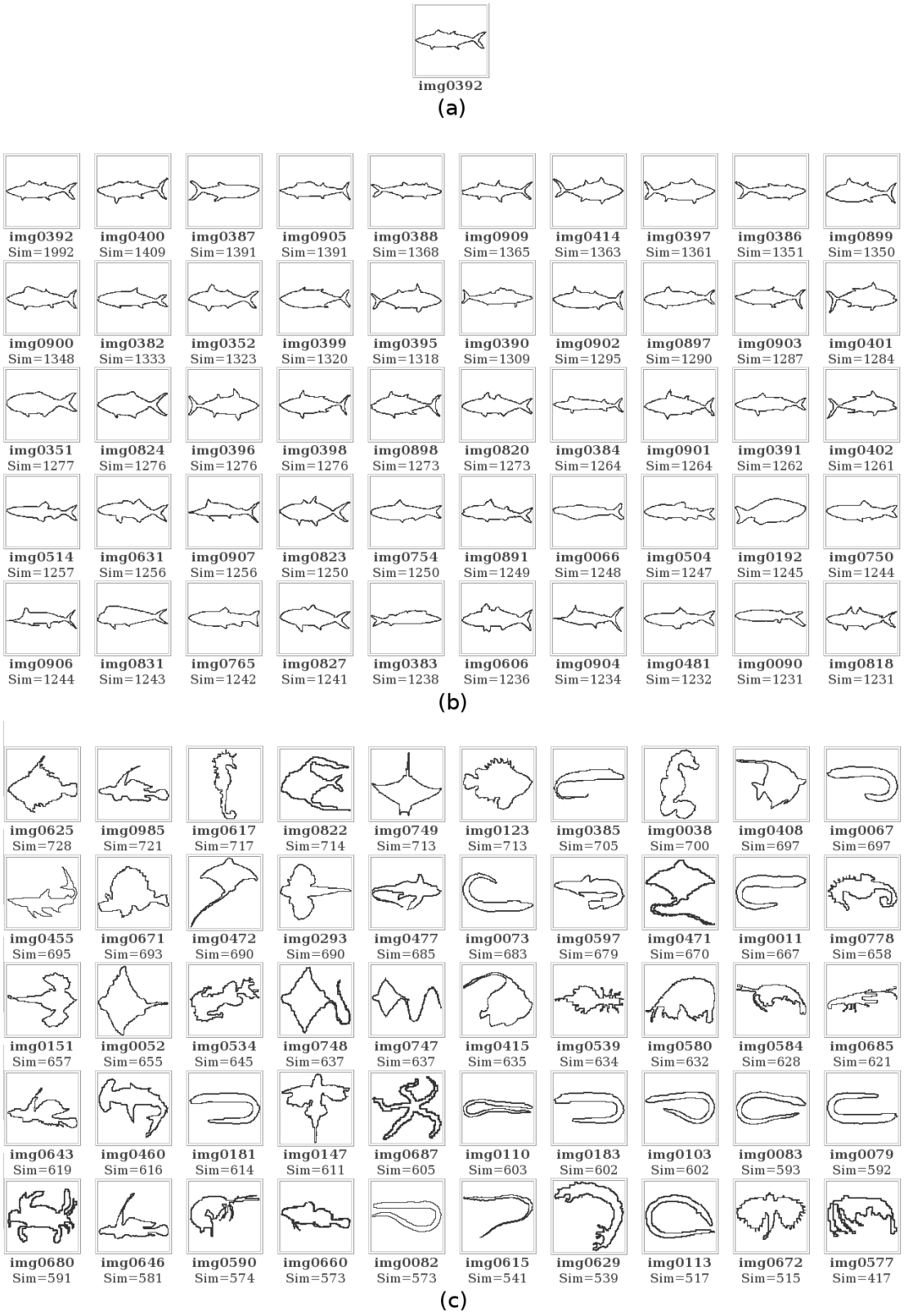


Figura 4.10: Similitud semiglobal con parámetros del Cuadro 4.1. (a) *img0392* a comparar; (b) las 50 formas de mayor similitud; (c) las 50 formas de menor similitud

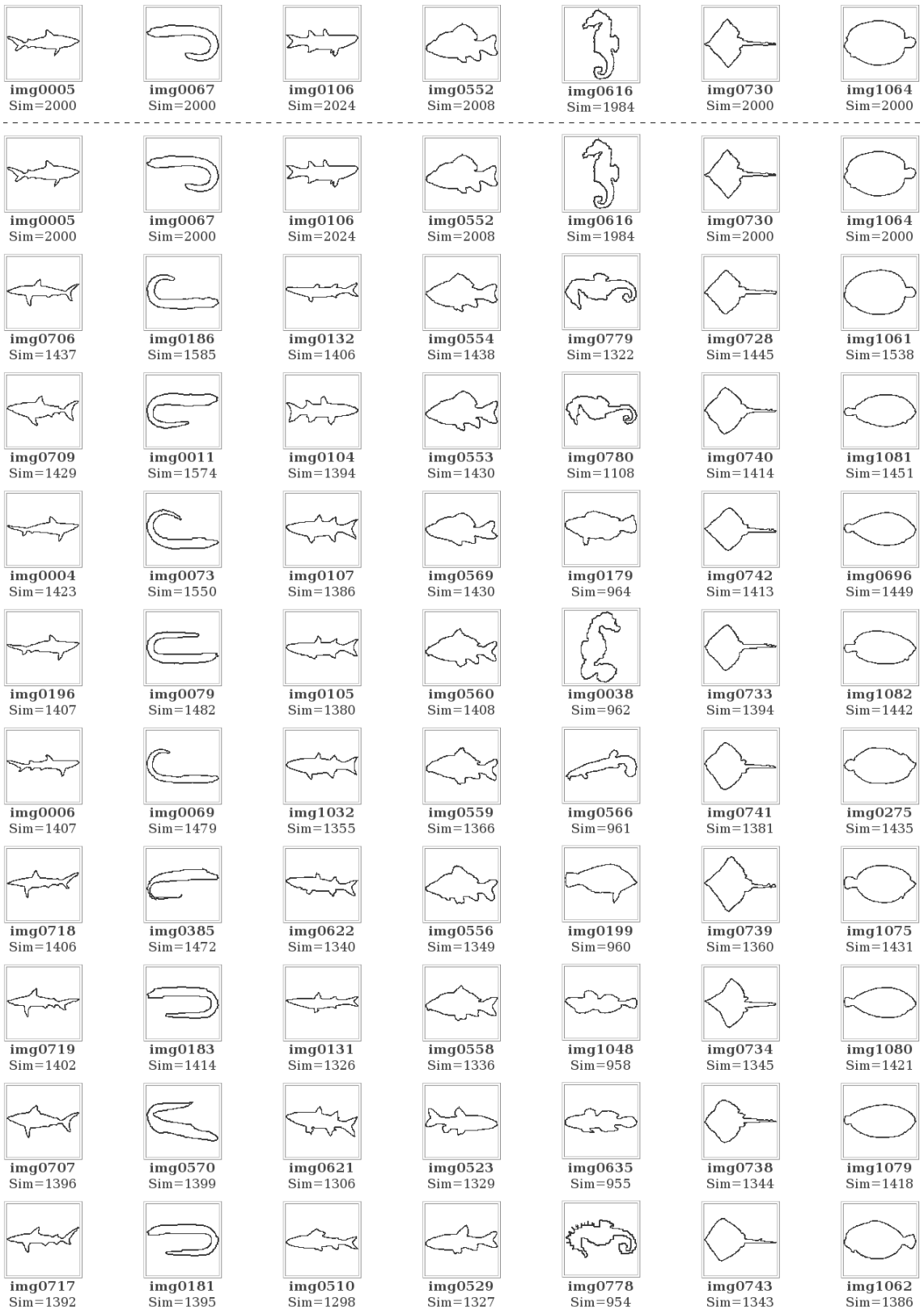


Figura 4.11: Similitud semiglobal con parámetros del Cuadro 4.1. Sobre la línea punteada se encuentra la entrada dada y debajo las 10 imágenes de mayor similitud (ordenadas de mayor a menor)

4.6. Consideraciones finales

Para terminar, se destacan algunas recomendaciones importantes. Siempre es recomendable empezar con la selección de un código cadena esperando que este sea robusto y que permita hacer fácilmente transformaciones a la imagen, ya que en muchos casos, hay que encontrar la similitud no solo con la forma discreta que representa, sino que incluso con el simétrico de esa forma discreta o probablemente con alguna de sus rotaciones.

Para el ejemplo concreto de las especies marinas se decidió usar el código *SACC* en su primer nivel de anidamento y trabajar así con un alfabeto finito (1, 2, 3 y 4) haciendo también muy simple el proceso de buscar en toda la cadena y sin importar el punto de inicio que se seleccione. Posiblemente en otro caso se desearía usar *SACC* considerando un punto de inicio conocido y revisando a detalle diferencias en segmentos con diferente anidamiento. Tal vez en esos casos no se probarían todas las rotaciones.

Finalmente, hay que tener claro que para una persona, el que dos imágenes sean parecidas entre sí es una decisión tomada por un proceso de mera inspección visual sin mayor herramienta que el ojo para la adquisición de la imagen y del cerebro para la interpretación de ella. Aunque en ocasiones esto se presta a subjetividades, en este trabajo se plantea que el tipo de respuestas que usualmente daría una persona, son el tipo de respuestas que se espera de una computadora cuando se le pide comparar formas discretas. Por esto, hay que tener cuidado con las condiciones en las que se pone a la computadora a hacer dichas comparaciones.

Una persona comúnmente decide que dos formas son parecidas haciendo un análisis a muy baja resolución. Es decir, de una forma detallada extrae las características más significativas y las compara con las características significativas de otra forma. A partir de esto, se sugiere entonces que para esos casos hay que trabajar con imágenes a baja resolución.

Esto no descarta que en ocasiones convenga comparar imágenes a alta resolución

para ciertas aplicaciones, simplemente hace ver que para encontrar la similitud entre formas discretas en un rango aproximado al que lo haría una persona, es importante considerar a que resolución trabajar las imágenes. De hecho, puede haber aplicaciones que por el contrario, comparen formas muy parecidas entre sí y que para poder encontrar las similitudes y diferencias entre ellas se requiera de imágenes con una resolución lo suficientemente alta para extraer los detalles más finos y que no fácilmente vería una persona en condiciones promedio.

Capítulo 5

Conclusiones

Se mostró que usando algoritmos de similitud entre cadenas para códigos de cadena que representan formas discretas es posible obtener una buena aproximación al resultado de similitud entre formas discretas.

Esta solución satisface las necesidades de representar de manera compacta y fácil de procesar una base de datos de formas. En el caso del almacenamiento, el peor de los casos para representar un objeto con un código de cadena que asigna un elemento por vértice, no representa mayor costo que el almacenamiento de la imagen completa como un mapa de bits. Por su parte, los algoritmos de comparación de cadenas basados en Programación Dinámica, en su forma más pura tienen una complejidad $O(n_X n_Y)$ donde n_X y n_Y son la cantidad de elementos en las cadenas que se comparan. Dada la sugerencia de trabajar con cadenas de longitud normalizada sabemos entonces que el orden de los algoritmos en su tiempo de ejecución es $O(n^2)$ pero que en algunos casos se puede mejorar planteando algunas restricciones al problema (subsecciones 3.3.4.3 y 3.3.4.4).

Otro tipo de información también puede ser obtenida de esta idea, como el de encontrar solamente un segmento de alta similitud entre cadenas usando la medida de similitud local.

Los mejores resultados fueron obtenidos usando el cálculo de la similitud semiglobal y teniendo en los parámetros de comparación un mecánismo para calibrar

la sensibilidad a los resultados esperados. Los parámetros usados fueron $M = 1$, $m = -1$ y $g = -2$ primeramente, y posteriormente se presentaron resultados usando los parámetros del Cuadro 4.1 habiendo antes explicado las dos posibles formas de representar dichos parámetros (sección 4.5).

Otro punto importante para manejar la sensibilidad de los resultados obtenidos es establecer el nivel de detalle que se desea considerar en la obtención de la similitud. Dado que en un análisis de forma regularmente se desea que los resultados de una computadora se asemejen a los de una persona, es importante entender que el análisis de una persona es hecho a baja resolución y sin tanto detalle, por lo que en ocasiones habrá que escalar las formas discretas para representarlas con una menor cantidad de elementos en el código cadena.

Con el uso de la similitud semiglobal y con algunos algoritmos de transformación fue posible obtener una medida de similitud adecuada y que es robusta hacía una serie de problemas como el escalamiento (solucionado con la normalización de cadenas), la orientación (solucionada con un algoritmo de rotación de cadenas), la simetría (solucionada con un algoritmo de transformación de reflejo) y el punto de incio (solucionado encontrando el mejor alineamiento con una de las dos cadenas duplicadas).

Para ejemplificar la medida de similitud, se uso una base de datos con 1100 formas discretas representando distintas especies marinas, que tras pasar por un proceso de binarización y extracción del código *SACC*, se normalizaron esas cadenas a una cantidad de elementos cercana a 500.

También, un nuevo código de cadena fue propuesto y una de sus variantes fue usada con éxito en la obtención de la medida de similitud. El nuevo código se basa en la pendiente acumulada desde un punto de inicio hacía cualquier vértice en el contorno del objeto y puede dar información importante en la representación de formas discretas orientadas. La primer derivada del código da información de formas sin considerar la orientación que estas formas discretas tengan.

Bibliografía

[1] Bribiesca E, “A new chain code”, PATTERN RECOGNITION, Volume 32, Issue 2, 235-251 (1999)

[2] Bribiesca E, “A Geometric structure for two-dimensional shapes and three-dimensional surfaces", PATTERN RECOGNITION, Volume 25, Issue 5, 483-496 (1992)

[3] Freeman H, “On the encoding of arbitrary geometric configurations”, IRE Trans EC-10 (2), 260-268 (1961)

[4] Petrakis EGM, Diplaros A, Milios E, “Matching and retrieval of distorted and occluded shapes using dynamic programming”, IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, Volume 24, Issue 11, 1501-1516 (2002)

[5] Bribiesca E, Wilson RG, ”A measure of 2D shape-of-object dissimilarity”, APPLIED MATHEMATICS LETTERS, Volume 10, Issue 6, 107-115 (1997)

[6] Gonzalez RC, Woods RE, “Digital Image Processing”, Prentice Hall, Third Edition, 640-678 (2007)

[7] Cormen T, Leiserson C, Rivest R, Stein C, “Introduction to Algorithms”, McGraw-Hill Science/Engineering/Math, Second Edition, 350-356 (2003)

[8] Navarro G, “A guided tour to approximate string matching”, ACM COMPUTING SURVEYS, Volume 33, Issue 1, 31-88 (2001)

[9] Levenshtein V, "Binary codes capable of correcting deletions, insertions and reversals", SOV. PHYS. DOKL. 10, 8, 707–710. Original in Russian in DOKL. AKAD. NAUK SSSR 163, 4, 845–848 (1965)

[10] Gusfield D, "Algorithms on strings, trees, and sequences: computer science and computational biology". Cambridge University Press, 215-253 (1997).

[11] Needleman SB, Wunsch CD, "A general method applicable to search for similarities in the amino acid sequence of two proteins", JOURNAL OF MOLECULAR BIOLOGY, 48, 443-453 (1970)

[12] Smith TF, Waterman MS, "Identification of common molecular subsequences", JOURNAL OF MOLECULAR BIOLOGY, 147, 195-197 (1981)

[13] Setubal C, Meidanis J, "Introduction to Computational Molecular Biology", PWS Publishing, First Edition, 33-103 (1997)

[14] `http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/a/Abbasi:Sadegh.html`

[15] `http://uxmcc3.iimas.unam.mx/elemus/similarity2d`

Apéndice A

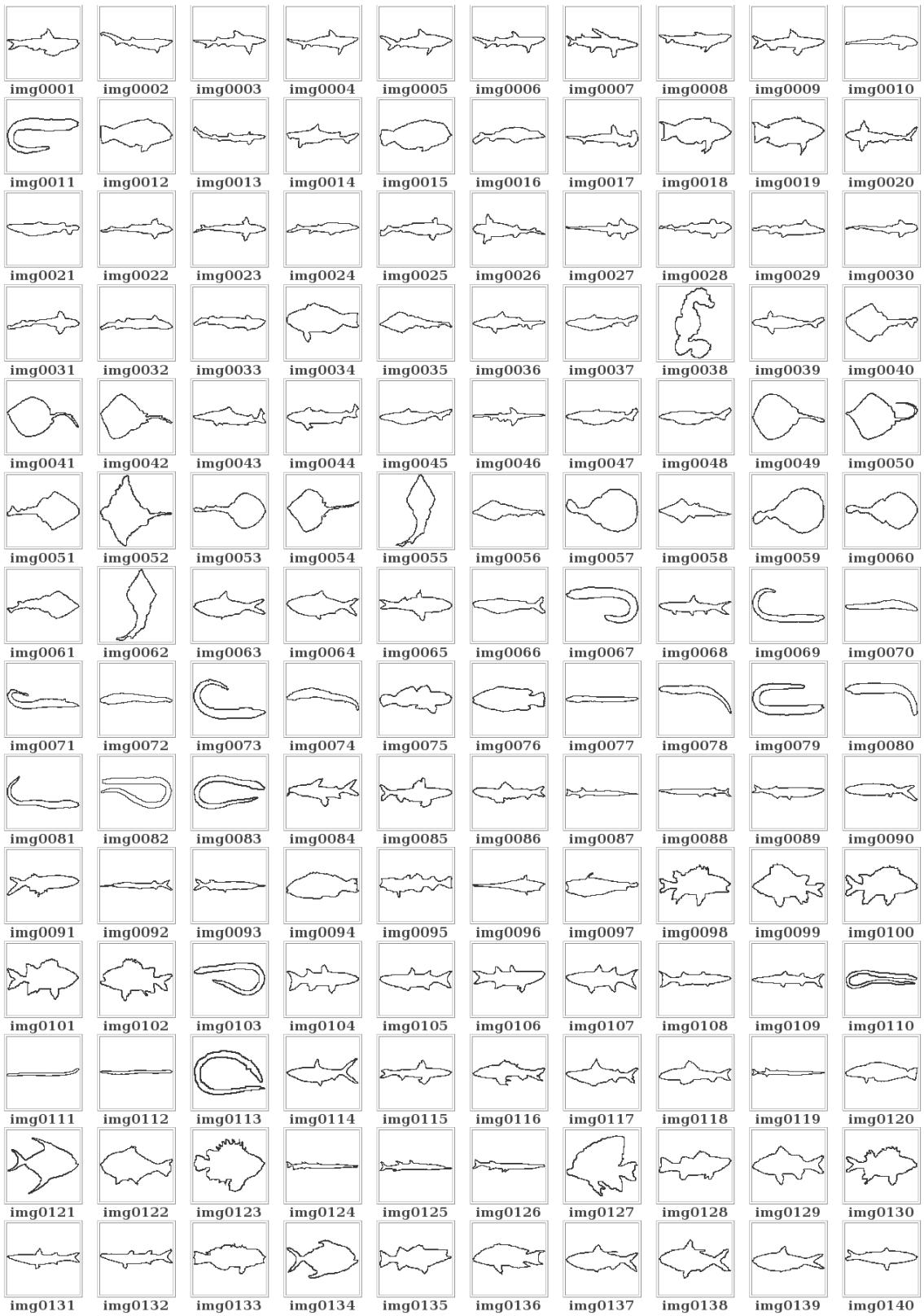
Base de datos de especies marinas[1]

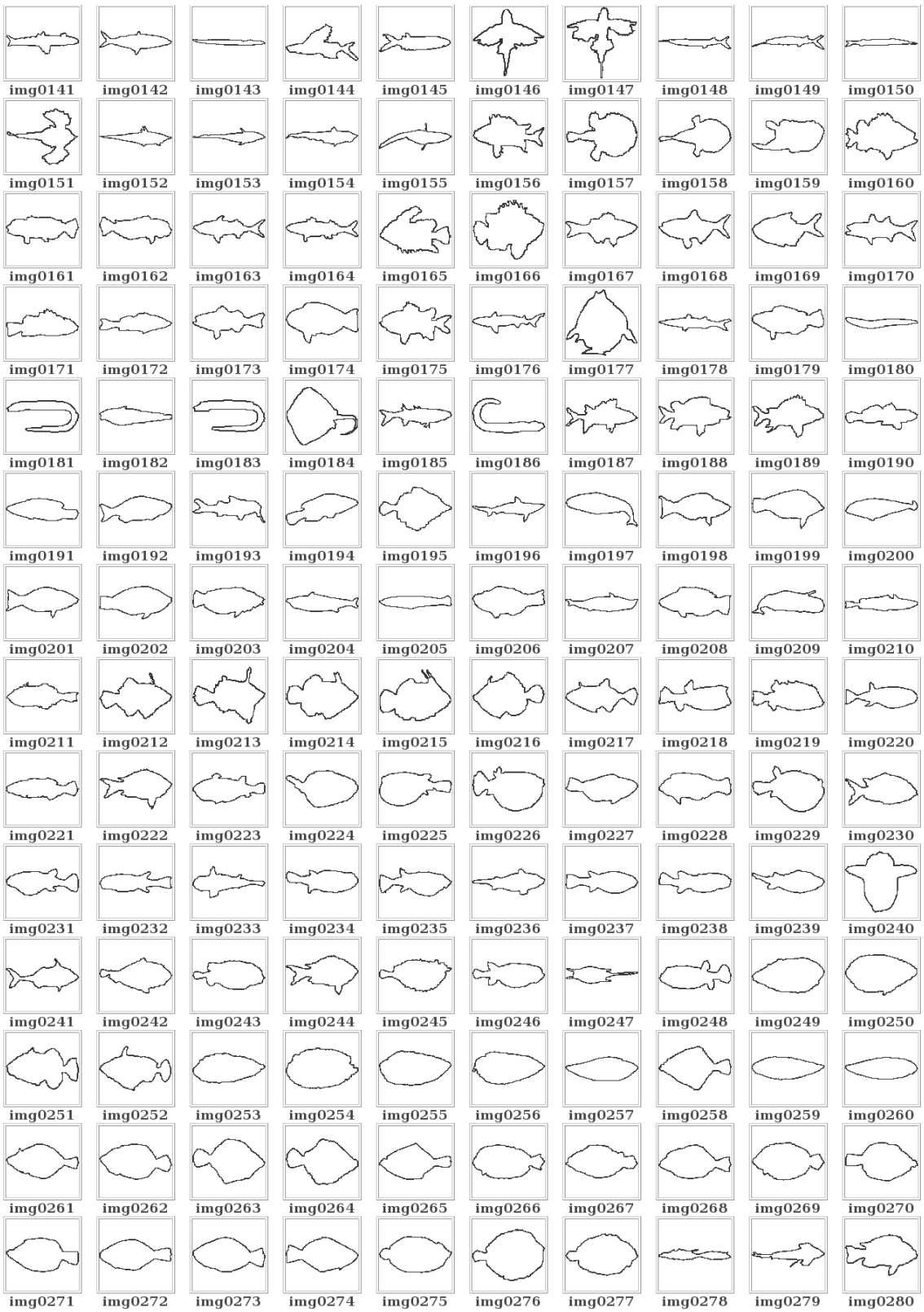
Con la finalidad de probar la extracción de distintos códigos cadena, de aplicar algoritmos de transformación y de analizar medidas de similitud, una base de datos de imágenes es necesaria. Para el análisis de formas que se está presentando se espera que esas imágenes sean imágenes binarias y que el objeto que en ellas aparece halla sido previamente segmentado aislando así otros problemas cuya solución no son la finalidad de lo que aquí se presenta.

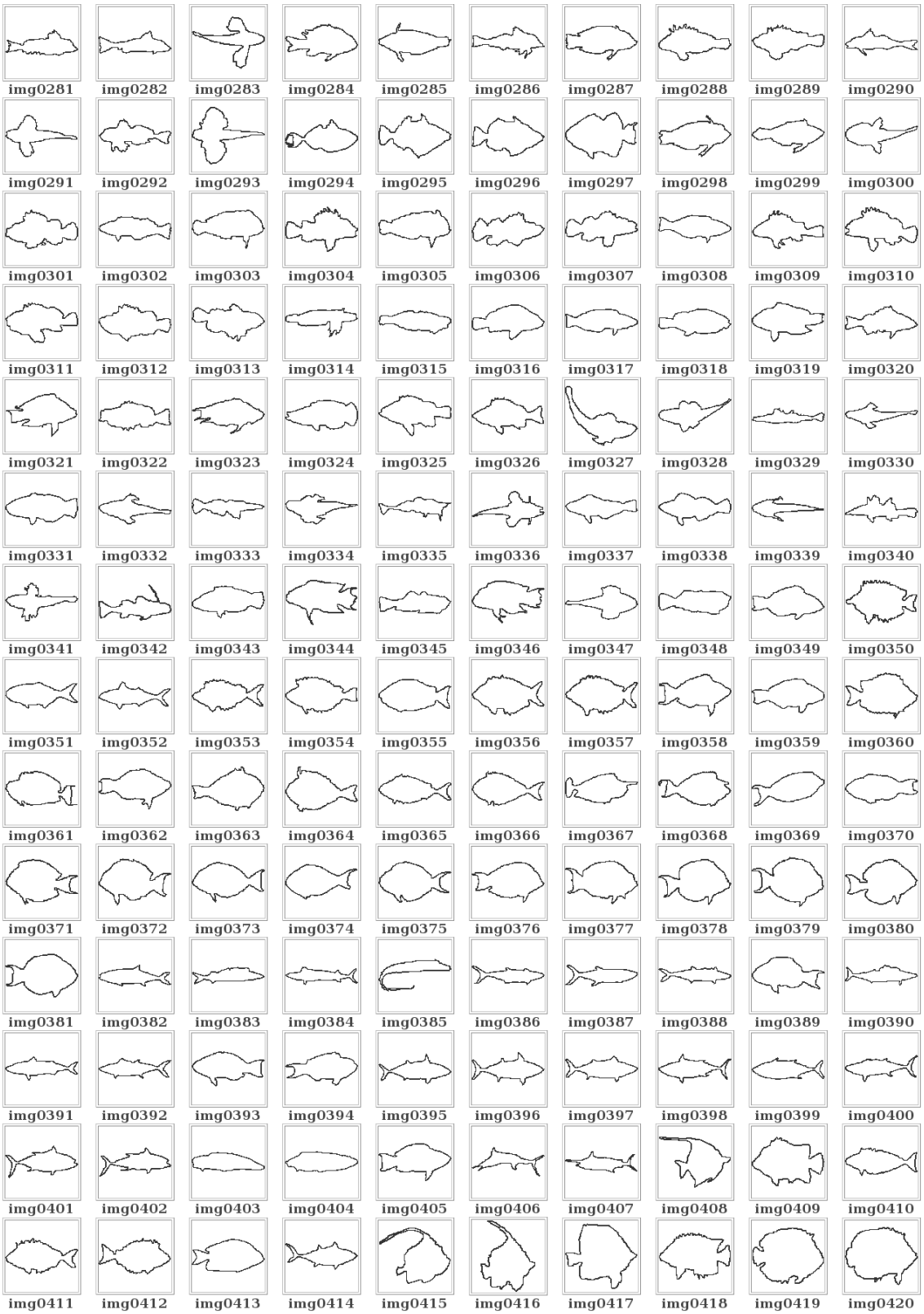
La base de datos usada [14] presenta 1100 formas discretas representando distintas especies marinas. Una vez obtenida esa base de datos, las imágenes pasaron por un proceso de binarización y extracción del código $SACC$, el cual fue normalizado con un factor $F = 500$ escalando las formas discretas lo necesario. El contorno que resulta de dicha transformación es con el que se generon las imágenes que se muestran a continuación y con las que se estará trabajando.

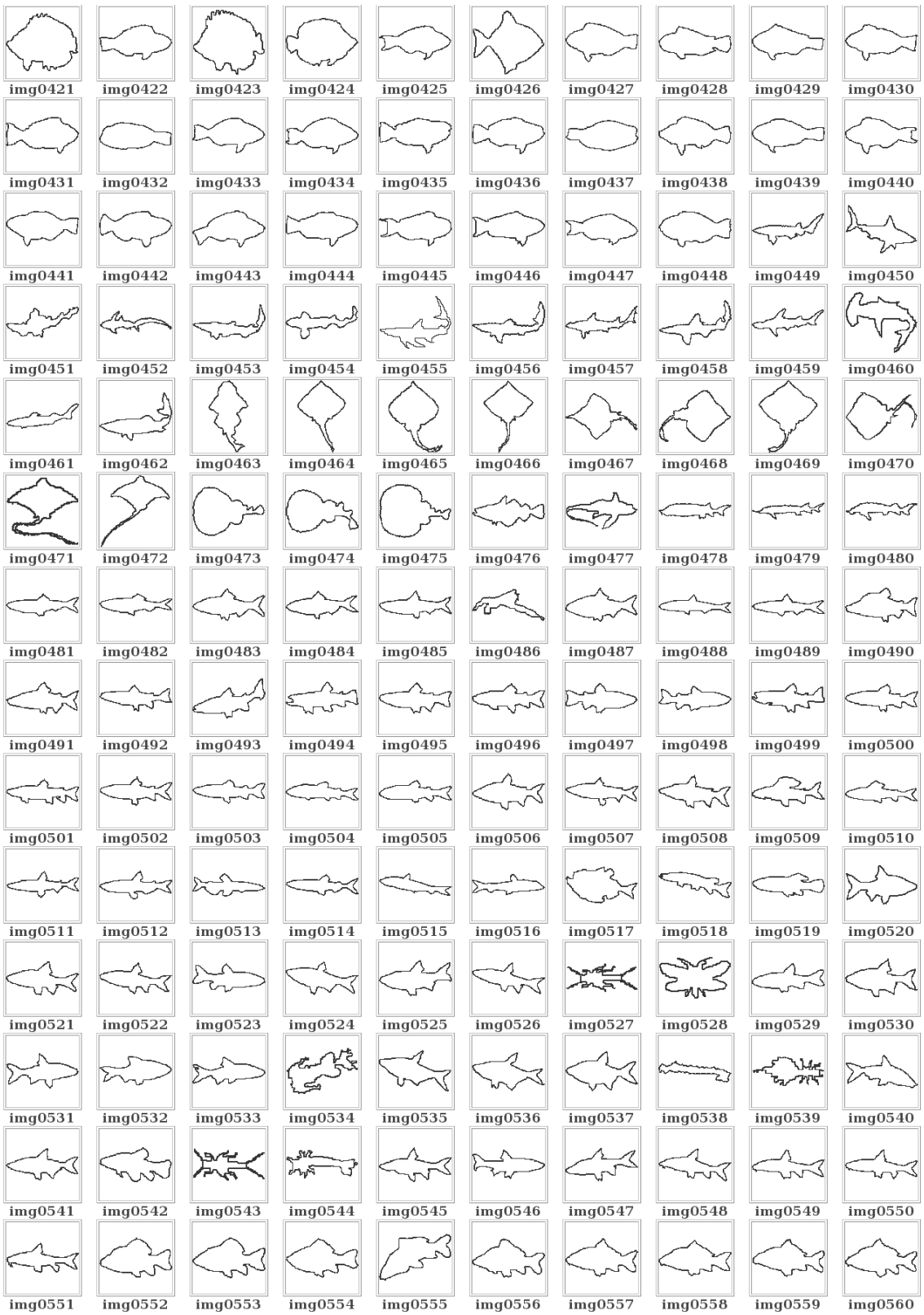
Por cuestiones de espacio y de presentación del documento, las imágenes no se muestran en su tamaño original.

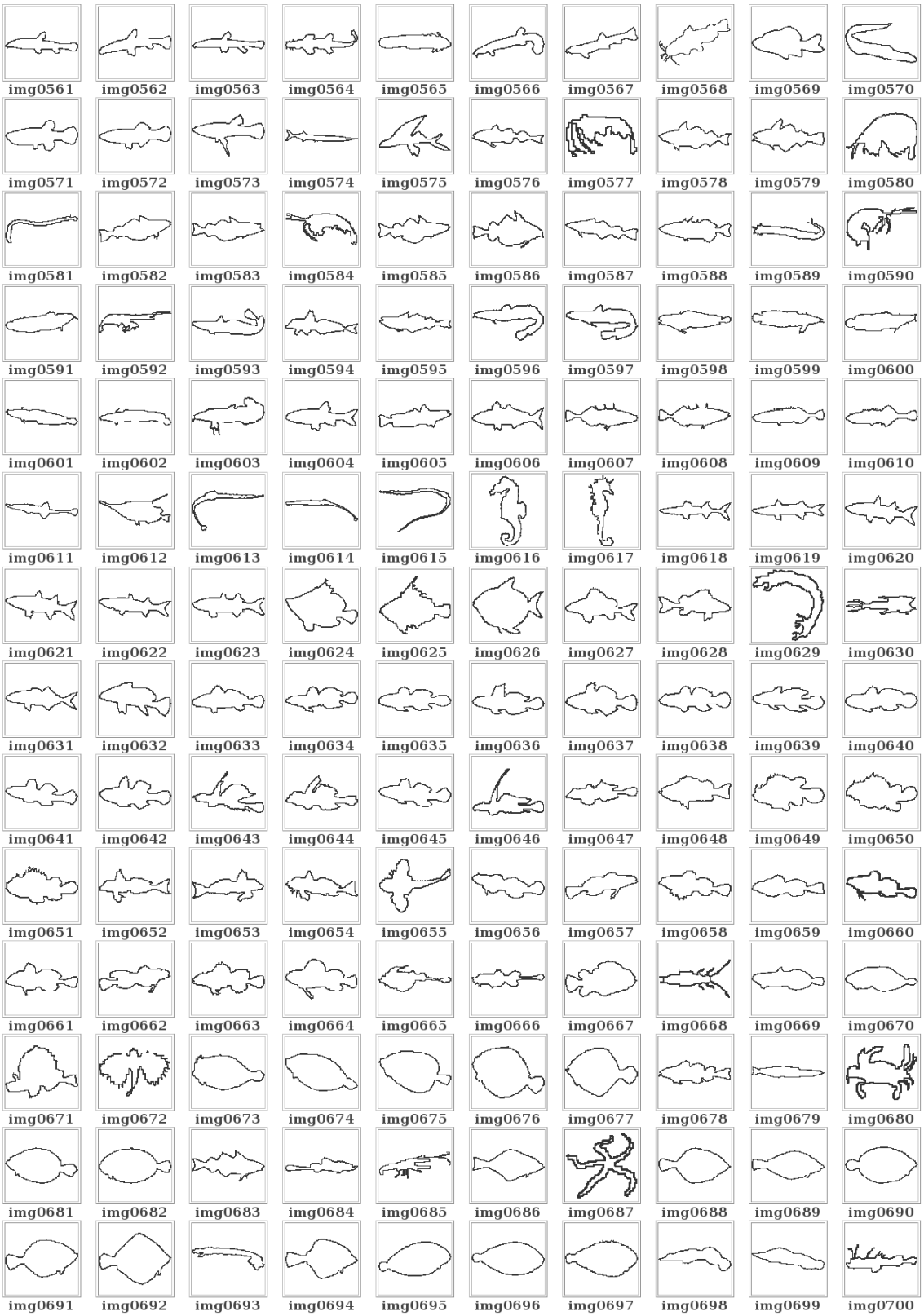
[1]Proporcionada por Sadegh Abbasi (`http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/a/Abbasi:Sadegh.html`)

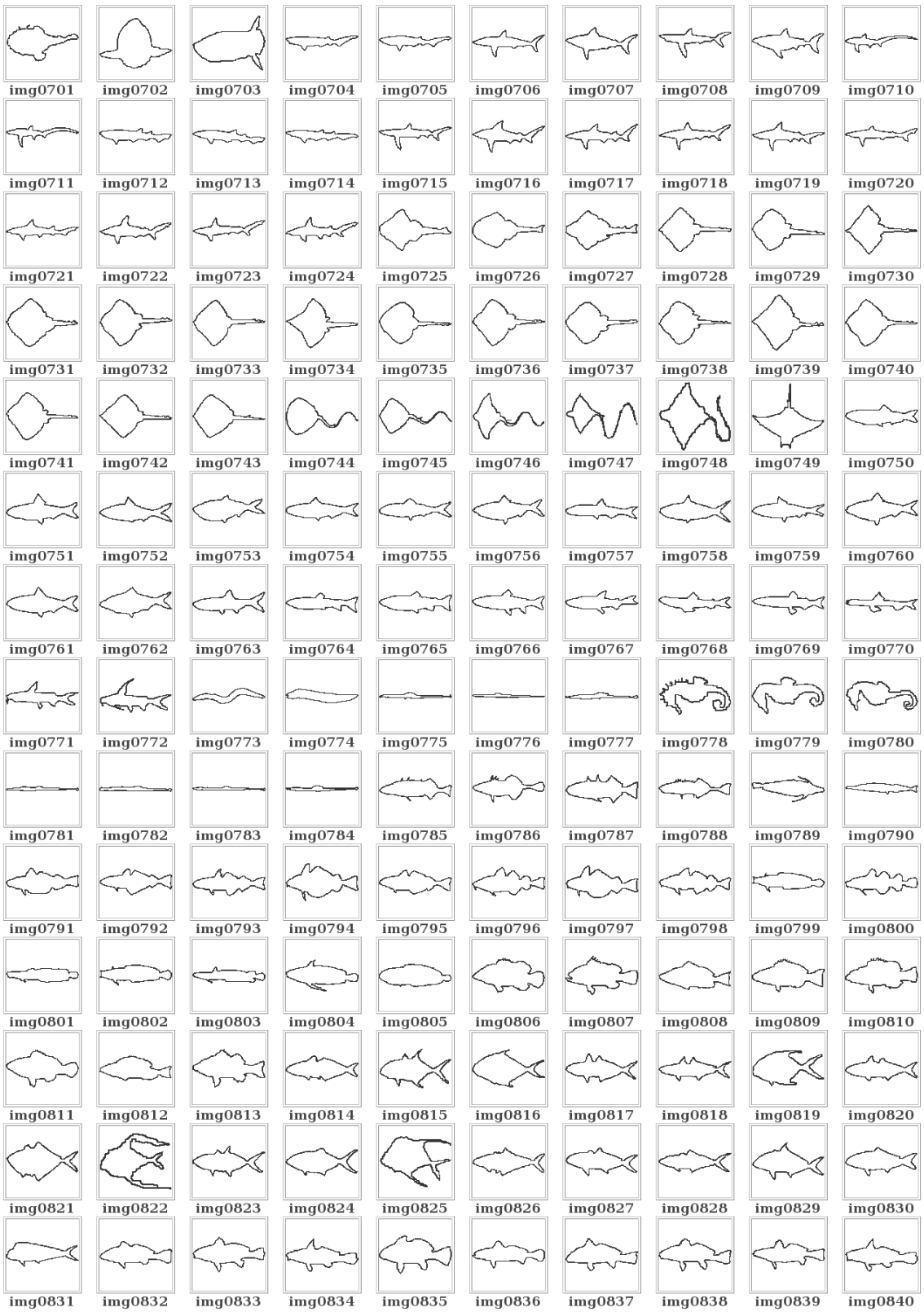

img0001 img0002 img0003 img0004 img0005 img0006 img0007 img0008 img0009 img0010
img0011 img0012 img0013 img0014 img0015 img0016 img0017 img0018 img0019 img0020
img0021 img0022 img0023 img0024 img0025 img0026 img0027 img0028 img0029 img0030
img0031 img0032 img0033 img0034 img0035 img0036 img0037 img0038 img0039 img0040
img0041 img0042 img0043 img0044 img0045 img0046 img0047 img0048 img0049 img0050
img0051 img0052 img0053 img0054 img0055 img0056 img0057 img0058 img0059 img0060
img0061 img0062 img0063 img0064 img0065 img0066 img0067 img0068 img0069 img0070
img0071 img0072 img0073 img0074 img0075 img0076 img0077 img0078 img0079 img0080
img0081 img0082 img0083 img0084 img0085 img0086 img0087 img0088 img0089 img0090
img0091 img0092 img0093 img0094 img0095 img0096 img0097 img0098 img0099 img0100
img0101 img0102 img0103 img0104 img0105 img0106 img0107 img0108 img0109 img0110
img0111 img0112 img0113 img0114 img0115 img0116 img0117 img0118 img0119 img0120
img0121 img0122 img0123 img0124 img0125 img0126 img0127 img0128 img0129 img0130
img0131 img0132 img0133 img0134 img0135 img0136 img0137 img0138 img0139 img0140

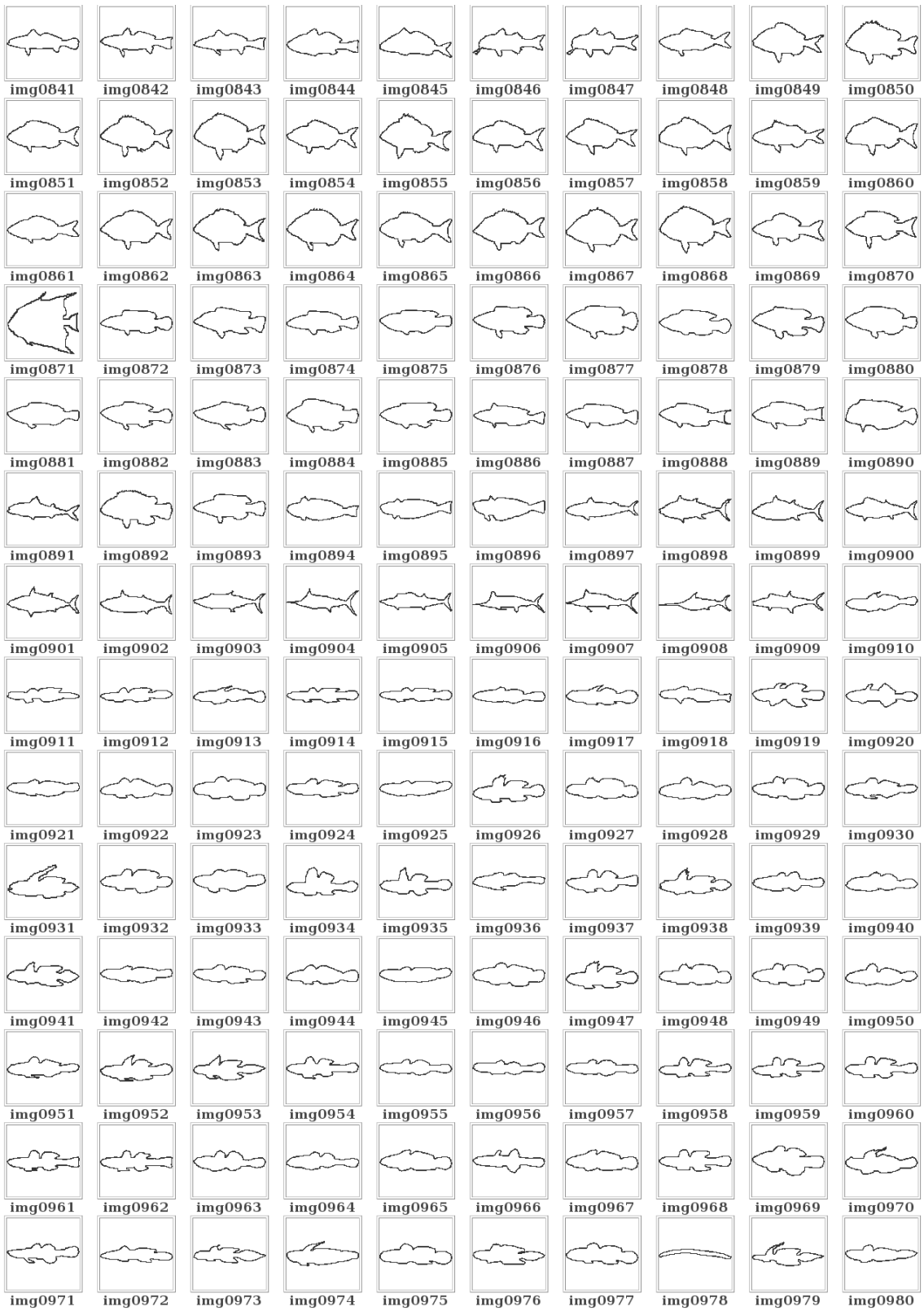

img0141 img0142 img0143 img0144 img0145 img0146 img0147 img0148 img0149 img0150
img0151 img0152 img0153 img0154 img0155 img0156 img0157 img0158 img0159 img0160
img0161 img0162 img0163 img0164 img0165 img0166 img0167 img0168 img0169 img0170
img0171 img0172 img0173 img0174 img0175 img0176 img0177 img0178 img0179 img0180
img0181 img0182 img0183 img0184 img0185 img0186 img0187 img0188 img0189 img0190
img0191 img0192 img0193 img0194 img0195 img0196 img0197 img0198 img0199 img0200
img0201 img0202 img0203 img0204 img0205 img0206 img0207 img0208 img0209 img0210
img0211 img0212 img0213 img0214 img0215 img0216 img0217 img0218 img0219 img0220
img0221 img0222 img0223 img0224 img0225 img0226 img0227 img0228 img0229 img0230
img0231 img0232 img0233 img0234 img0235 img0236 img0237 img0238 img0239 img0240
img0241 img0242 img0243 img0244 img0245 img0246 img0247 img0248 img0249 img0250
img0251 img0252 img0253 img0254 img0255 img0256 img0257 img0258 img0259 img0260
img0261 img0262 img0263 img0264 img0265 img0266 img0267 img0268 img0269 img0270
img0271 img0272 img0273 img0274 img0275 img0276 img0277 img0278 img0279 img0280

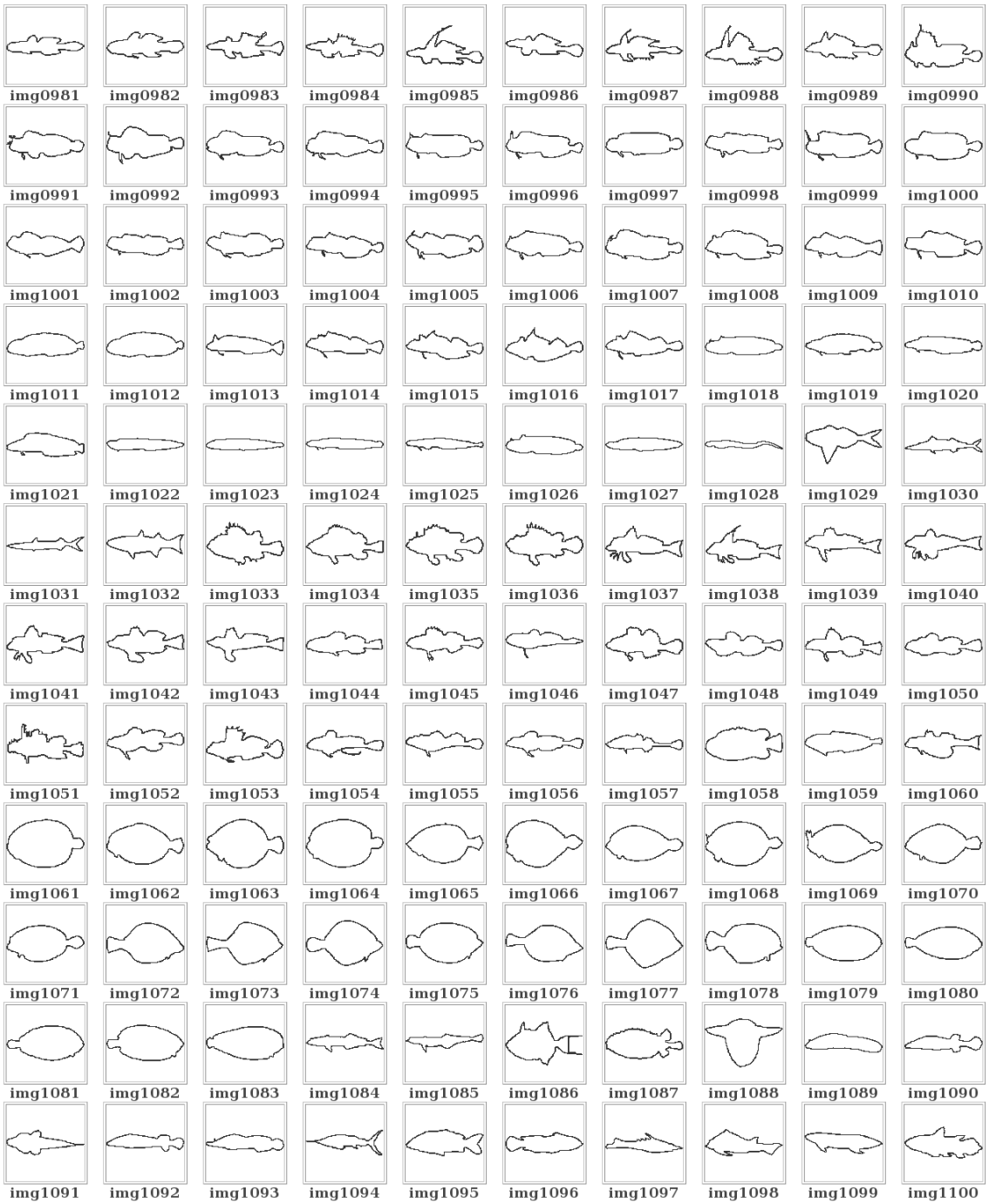

img0281 img0282 img0283 img0284 img0285 img0286 img0287 img0288 img0289 img0290
img0291 img0292 img0293 img0294 img0295 img0296 img0297 img0298 img0299 img0300
img0301 img0302 img0303 img0304 img0305 img0306 img0307 img0308 img0309 img0310
img0311 img0312 img0313 img0314 img0315 img0316 img0317 img0318 img0319 img0320
img0321 img0322 img0323 img0324 img0325 img0326 img0327 img0328 img0329 img0330
img0331 img0332 img0333 img0334 img0335 img0336 img0337 img0338 img0339 img0340
img0341 img0342 img0343 img0344 img0345 img0346 img0347 img0348 img0349 img0350
img0351 img0352 img0353 img0354 img0355 img0356 img0357 img0358 img0359 img0360
img0361 img0362 img0363 img0364 img0365 img0366 img0367 img0368 img0369 img0370
img0371 img0372 img0373 img0374 img0375 img0376 img0377 img0378 img0379 img0380
img0381 img0382 img0383 img0384 img0385 img0386 img0387 img0388 img0389 img0390
img0391 img0392 img0393 img0394 img0395 img0396 img0397 img0398 img0399 img0400
img0401 img0402 img0403 img0404 img0405 img0406 img0407 img0408 img0409 img0410
img0411 img0412 img0413 img0414 img0415 img0416 img0417 img0418 img0419 img0420


img0421 img0422 img0423 img0424 img0425 img0426 img0427 img0428 img0429 img0430
img0431 img0432 img0433 img0434 img0435 img0436 img0437 img0438 img0439 img0440
img0441 img0442 img0443 img0444 img0445 img0446 img0447 img0448 img0449 img0450
img0451 img0452 img0453 img0454 img0455 img0456 img0457 img0458 img0459 img0460
img0461 img0462 img0463 img0464 img0465 img0466 img0467 img0468 img0469 img0470
img0471 img0472 img0473 img0474 img0475 img0476 img0477 img0478 img0479 img0480
img0481 img0482 img0483 img0484 img0485 img0486 img0487 img0488 img0489 img0490
img0491 img0492 img0493 img0494 img0495 img0496 img0497 img0498 img0499 img0500
img0501 img0502 img0503 img0504 img0505 img0506 img0507 img0508 img0509 img0510
img0511 img0512 img0513 img0514 img0515 img0516 img0517 img0518 img0519 img0520
img0521 img0522 img0523 img0524 img0525 img0526 img0527 img0528 img0529 img0530
img0531 img0532 img0533 img0534 img0535 img0536 img0537 img0538 img0539 img0540
img0541 img0542 img0543 img0544 img0545 img0546 img0547 img0548 img0549 img0550
img0551 img0552 img0553 img0554 img0555 img0556 img0557 img0558 img0559 img0560


img0561 img0562 img0563 img0564 img0565 img0566 img0567 img0568 img0569 img0570
img0571 img0572 img0573 img0574 img0575 img0576 img0577 img0578 img0579 img0580
img0581 img0582 img0583 img0584 img0585 img0586 img0587 img0588 img0589 img0590
img0591 img0592 img0593 img0594 img0595 img0596 img0597 img0598 img0599 img0600
img0601 img0602 img0603 img0604 img0605 img0606 img0607 img0608 img0609 img0610
img0611 img0612 img0613 img0614 img0615 img0616 img0617 img0618 img0619 img0620
img0621 img0622 img0623 img0624 img0625 img0626 img0627 img0628 img0629 img0630
img0631 img0632 img0633 img0634 img0635 img0636 img0637 img0638 img0639 img0640
img0641 img0642 img0643 img0644 img0645 img0646 img0647 img0648 img0649 img0650
img0651 img0652 img0653 img0654 img0655 img0656 img0657 img0658 img0659 img0660
img0661 img0662 img0663 img0664 img0665 img0666 img0667 img0668 img0669 img0670
img0671 img0672 img0673 img0674 img0675 img0676 img0677 img0678 img0679 img0680
img0681 img0682 img0683 img0684 img0685 img0686 img0687 img0688 img0689 img0690
img0691 img0692 img0693 img0694 img0695 img0696 img0697 img0698 img0699 img0700


img0701 img0702 img0703 img0704 img0705 img0706 img0707 img0708 img0709 img0710
img0711 img0712 img0713 img0714 img0715 img0716 img0717 img0718 img0719 img0720
img0721 img0722 img0723 img0724 img0725 img0726 img0727 img0728 img0729 img0730
img0731 img0732 img0733 img0734 img0735 img0736 img0737 img0738 img0739 img0740
img0741 img0742 img0743 img0744 img0745 img0746 img0747 img0748 img0749 img0750
img0751 img0752 img0753 img0754 img0755 img0756 img0757 img0758 img0759 img0760
img0761 img0762 img0763 img0764 img0765 img0766 img0767 img0768 img0769 img0770
img0771 img0772 img0773 img0774 img0775 img0776 img0777 img0778 img0779 img0780
img0781 img0782 img0783 img0784 img0785 img0786 img0787 img0788 img0789 img0790
img0791 img0792 img0793 img0794 img0795 img0796 img0797 img0798 img0799 img0800
img0801 img0802 img0803 img0804 img0805 img0806 img0807 img0808 img0809 img0810
img0811 img0812 img0813 img0814 img0815 img0816 img0817 img0818 img0819 img0820
img0821 img0822 img0823 img0824 img0825 img0826 img0827 img0828 img0829 img0830
img0831 img0832 img0833 img0834 img0835 img0836 img0837 img0838 img0839 img0840


img0841 img0842 img0843 img0844 img0845 img0846 img0847 img0848 img0849 img0850
img0851 img0852 img0853 img0854 img0855 img0856 img0857 img0858 img0859 img0860
img0861 img0862 img0863 img0864 img0865 img0866 img0867 img0868 img0869 img0870
img0871 img0872 img0873 img0874 img0875 img0876 img0877 img0878 img0879 img0880
img0881 img0882 img0883 img0884 img0885 img0886 img0887 img0888 img0889 img0890
img0891 img0892 img0893 img0894 img0895 img0896 img0897 img0898 img0899 img0900
img0901 img0902 img0903 img0904 img0905 img0906 img0907 img0908 img0909 img0910
img0911 img0912 img0913 img0914 img0915 img0916 img0917 img0918 img0919 img0920
img0921 img0922 img0923 img0924 img0925 img0926 img0927 img0928 img0929 img0930
img0931 img0932 img0933 img0934 img0935 img0936 img0937 img0938 img0939 img0940
img0941 img0942 img0943 img0944 img0945 img0946 img0947 img0948 img0949 img0950
img0951 img0952 img0953 img0954 img0955 img0956 img0957 img0958 img0959 img0960
img0961 img0962 img0963 img0964 img0965 img0966 img0967 img0968 img0969 img0970
img0971 img0972 img0973 img0974 img0975 img0976 img0977 img0978 img0979 img0980


img0981 img0982 img0983 img0984 img0985 img0986 img0987 img0988 img0989 img0990
img0991 img0992 img0993 img0994 img0995 img0996 img0997 img0998 img0999 img1000
img1001 img1002 img1003 img1004 img1005 img1006 img1007 img1008 img1009 img1010
img1011 img1012 img1013 img1014 img1015 img1016 img1017 img1018 img1019 img1020
img1021 img1022 img1023 img1024 img1025 img1026 img1027 img1028 img1029 img1030
img1031 img1032 img1033 img1034 img1035 img1036 img1037 img1038 img1039 img1040
img1041 img1042 img1043 img1044 img1045 img1046 img1047 img1048 img1049 img1050
img1051 img1052 img1053 img1054 img1055 img1056 img1057 img1058 img1059 img1060
img1061 img1062 img1063 img1064 img1065 img1066 img1067 img1068 img1069 img1070
img1071 img1072 img1073 img1074 img1075 img1076 img1077 img1078 img1079 img1080
img1081 img1082 img1083 img1084 img1085 img1086 img1087 img1088 img1089 img1090
img1091 img1092 img1093 img1094 img1095 img1096 img1097 img1098 img1099 img1100

Apéndice B

Glosario

CAPÍTULO 2	Código Cadena de Pendiente Acumulada (SACC)
α	Ángulo de rotación.
θ_i	Ángulo de cambio en el i-ésimo vértice del contorno.
A	Cadena o secuencia ordenada de elementos.
A'	Primera derivada $SACC$ de una cadena A.
a_i	i-ésimo elemento de una cadena A.
$a_i^{'}$	i-esimo elemento de una cadena A'.
a_i^M	Reflejo horizontal del elemento a_i de una cadena A.
B_i	Cantidad de celdas del objeto en contacto con el i-ésimo vértice.
k	Número de incrementos a rotar derivado del ángulo de rotación α.
l	Longitud de cada lado de una celda regular (considerado ser 1).
L	Longitud de una cadena.
Mod	Operación módulo. Da el residuo de una división entera.
n	Número de elementos de una cadena.
N	Nivel de anidamiento.
o	Número de orientaciones para cualquier segmento del contorno.
$pos_X(a_i)$	Posición relativa al punto de inicio sobre el eje X para a_i.
$pos_X^*(a_i)$	Posición absoluta sobre el eje X para a_i.

q	Cantidad de elementos de un prefijo de la cadena A.
r	Factor de escalamiento en todos los ejes coordenados.
r_E	Factor de escalamiento en el eje coordenado E.
$sacc_i$	i-ésimo elemento de una cadena $SACC$.
$sacc_i^{'}$	i-ésimo elemento de una cadena $SACC'$.
$slope_i$	Incremento en la pendiente acumulada para el i-ésimo vértice.
T	Cantidad de celdas totales en contacto con cualquier vértice.
v	Un vértice sobre el contorno del objeto.
vcc_i	i-ésimo elemento de una cadena VCC.

CAPÍTULO 3	Algoritmos de semejanza entre cadenas
$D(\,)$	Función de distancia entre cadenas.
$D[\,]$	Matriz de almacenamiento para las salidas de la función D.
g	Costo asociado a una inserción / elimiación (*gap*).
I	Secuencia incremental de índices.
m	Costo asociado a un remplazo (*mismatch*).
M	Costo asociado a un emparejamiento (*match*).
n_X	Cantidad de elementos de la cadena X
n_Y	Cantidad de elementos de la cadena Y
$S(\,)$	Función de similitud entre cadenas.
$S[\,]$	Matriz de almacenamiento para las salidas de la función S.
S_G	Función / Matriz de similitud global.
S_L	Función / Matriz de similitud local.
S_S	Función / Matriz de similitud semiglobal.
t	Función de comparación de caracteres.
$T[\,]$	Matriz de rutas para *LCS*.
U	Universo de elementos para una cadena (Alfabeto).

X	Cadena a comparar con Y.
X^s	Subcadena de X.
Y	Cadena a comparar con X.
Y^s	Subcadena de Y.
Z	Subsecuenca común entre X y Y.

CAPÍTULO 4	Propuesta de un algoritmo de semejanza entre formas discretas
A	Área o cantidad de píxeles del objeto en una imagen.
F	Factor de normalización de una cadena.