UNIVERSITAT JAUME·I

Departament de LIenguatges i Sistemes Informàtics
Universitat Jaume I

Alineamiento de cadenas cíclicas

en el reconocimiento de formas bidimensionales

Tesis doctoral

Vicente Palazón González

Dirigida por el Dr. Andrés Marzal Varó

Castellón, mayo de 2010

A la memoria de mi padre.
A mi madre, abuelita y hermano.

Resumen

Cuando queremos comparar dos formas bidimensionales utilizando sus contornos, suele presentarse un problema importante: la invarianza al punto inicial en su codificación como secuencia. Aunque existen métodos heurísticos para conseguir un buen punto de inicio que funcionan en ciertos contextos, si queremos una solución genérica, la única manera de conseguir esta invarianza es midiendo distancias con todos los posibles puntos iniciales, es decir, utilizando el alineamiento por fuerza bruta con todo posible inicio de la secuencia del contorno. De aquí surge el concepto de cadena cíclica. Así, medir una distancia entre dos cadenas cíclicas sería lo mismo que medir una distancia entre todos los posibles puntos iniciales de las dos cadenas. Esta comparación es muy costosa computacionalmente y el trabajo de la literatura se ha orientado sobre todo a reducir este coste.

Existe mucho trabajo, a este respecto, en el dominio de las distancias de edición. Sin embargo, con otras técnicas, como son el alineamiento temporal no lineal (en inglés, *Dynamic Time Warping*) o los modelos ocultos de Markov (más tolerantes al ruido y otras deformaciones), no se ha profundizado demasiado con las cadenas cíclicas. Las aportaciones de esta tesis, van orientadas en esta dirección.

Con el alineamiento temporal no lineal (ATNL), hemos desarrollado un algoritmo eficiente para el cálculo del ATNL cíclico. Hemos planteado también diversas alternativas para acelerar el cálculo del ATNL cíclico en tareas de reconocimiento. En primer lugar, un heurístico para evitar el cálculo cíclico, en el caso de que tengamos categorías etiquetadas. En segundo lugar, un método óptimo para acelerar el cálculo cíclico, utilizando una cota inferior basada en un pseudo-alineamiento que aproxima la distancia cíclica. Finalmente, aportamos soluciones basadas en AESA (*Approximating and Eliminating Search Algorithm*) y una mejora al algoritmo LAESA (*Linear* AESA).

Con los modelos ocultos de Markov, estudiamos la topología lineal en el reconocimiento de contornos y desarrollamos extensiones cíclicas para los algoritmos de Viterbi (reconocimiento y entrenamiento) y Baum-Welch (entrenamiento).

Agradecimientos

En primer lugar, expresar mi agradecimiento a todos los integrantes del grupo ACRATA, en especial, a mi director de tesis, Andrés Marzal, por su ánimo, ayuda y paciencia, a Juan Miguel Vilar por su ayuda incondicional, y a Antonio Castellanos y Juan Carlos Amengual por su compañía.

Mi agradecimiento también a los integrantes del grupo de Visión, en especial a Pedro García, a Filiberto Pla y a José Martínez, y a los del *Temporal Knowledge Bases Group*, en particular a Rafael Berlanga. Siempre han estado ahí cuando les he necesitado.

En general, gracias a los miembros del departamento de lenguajes y sistemas informáticos que en mayor o menor medida me han ayudado.

Por último, gracias a las personas que contribuyeron a que superara los malos momentos.

En el plano institucional, agradecimientos para el Ministerio de Educación y Ciencia por la financiación del proyecto TIC2002-02684 (MATRICS) y la concesión de una beca FPI BES-2003-1670. Agradecimientos al Ministerio de Ciencia e Innovación por la financiación de los proyectos TIN2006-12767 (STATE) y CSD2007-00018 (MIPRCV), este último dentro del programa Consolider-Ingenio 2010. Agradecimientos también para la Universitat Jaume I por la concesión de una beca de continuidad en la actividad investigadora.

Índice general

Capítulo 1

Introducción

Ferrarin: «*There are warrants out on you for treason, illegal entry, decadence, pornography... and for being a lazy pig.*»
Hayao Miyazaki, *Pòrco Rósso* (1998).

En este capítulo se introducirán generalidades sobre el reconocimiento de patrones centrándonos en el reconocimiento de formas bidimensionales y en el tema principal de la tesis: el alineamiento de cadenas cíclicas.

1.1. Reconocimiento de patrones

Una gran cantidad de nuestro conocimiento sobre el mundo que nos rodea está basado en patrones complejos: textos, música, piezas industriales, rostros, etc. Para la psicología, el problema fundamental en el ámbito del reconocimiento de patrones es el estudio de los mecanismos por los que las señales externas estimulan los órganos sensoriales y se convierten en experiencias perceptuales significativas. Estos procesos continúan siendo desconocidos en su mayor parte y no se ha encontrado un modelo concluyente sobre cómo nuestro sistema nervioso realiza este reconocimiento. Sin embargo, se admite que esta tarea debe realizarse siguiendo un esquema general como el que se detalla a continuación. Antes del reconocimiento, un patrón debe ser percibido por los órganos sensoriales. Además, el mismo patrón o alguno similar (de la misma categoría) debe haberse percibido y recordado previamente. Finalmente, debe establecerse alguna correspondencia entre la percepción actual y lo recordado.

El uso intensivo de ordenadores y otros dispositivos electrónicos en los últimos años ha impulsado el estudio y aplicación de técnicas de reconocimiento de patrones. En particular, el estudio de las teorías y técnicas de reconocimiento implementables en un sistema informático.

El reconocimiento de patrones (RP) es el estudio de cómo las máquinas pueden observar el entorno, aprender a distinguir patrones de interés y realizar decisiones acertadas según las categorías de estos patrones. En este sentido, son numerosas las fuentes de este área:

las Matemáticas, la Estadística, la Informática, etc. Las aplicaciones tienen lugar en muy diversos campos: Biología, Medicina, Psicología, Economía, Ingeniería, etc.

Aunque la aplicabilidad de las técnicas resulta, a priori, muy amplia, no hay un método que sea la panacea. Existen diversas razones que hacen que los sistemas de RP operativos sean muy específicos respecto al problema a resolver:

- La naturaleza de los patrones: caracteres escritos, símbolos, dibujos, imágenes biomédicas, firmas, objetos tridimensionales, huellas dactilares, espectrogramas, cromosomas, etc.
- Los requerimientos del sistema: especialmente en tiempo de respuesta, que hace que algunos métodos de reconocimiento, aun siendo superiores en éxito, no sean aplicables en la práctica.
- Factores económicos: un sistema equipado con diferentes sensores y equipos de procesamiento muy potentes puede dar resultados muy satisfactorios y no ser asumibles por los usuarios dado su coste.

Estos factores hacen que un sistema adecuado para un problema sea inaplicable para otro, lo que alienta el estudio y desarrollo de nuevas técnicas.

1.2. Configuración de un sistema reconocedor

La configuración típica de un sistema reconocedor concreta las siguientes etapas:

- Adquisición: las magnitudes físicas (por ejemplo, imágenes) se transforman mediante sensores en señales eléctricas que una vez filtradas, amplificadas y digitalizadas pueden procesarse en el computador.
- Preprocesamiento: a menudo es conveniente mejorar la calidad de los datos originales (por ejemplo, para eliminar ruido o entidades irrelevantes, para ajustar el contraste). En otros casos interesa transformarlos a una representación más adecuada para su tratamiento matemático, con menor redundancia y/o con un realce de aspectos relevantes para la segmentación (la siguiente etapa).
- Segmentación: sirve para encontrar dentro del flujo de información los elementos individuales que parecen estar dotados de significado. La segmentación puede ser de tipo temporal, cuando se aislan fragmentos de la cadena de datos sensoriales recibidos a lo largo del tiempo (por ejemplo, imágenes de vídeo), o espacial, cuando el conjunto de datos recibidos en un cierto instante contiene varias posibles unidades relevantes (por ejemplo, formas u objetos de una escena o partes de un objeto). Excepto en situaciones triviales en las que el fondo y la forma están claramente diferenciados dentro de la masa de estímulos, la segmentación es en realidad un problema tan complejo como la propia interpretación de la escena, donde pueden existir problemas por ocultaciones y solapamientos parciales, deformaciones, no uniformidad en

iluminación, color u otras propiedades visuales, pseudo-objetos irrelevantes, etc. En el marco de la búsqueda de formas u objetos en escenas las técnicas más utilizadas son la búsqueda de regiones conexas y los contornos activos.

- Extracción de propiedades: en esta etapa se calculan propiedades (atributos, características) de cada entidad segmentada que deben ser, idealmente, discriminantes de las diferentes clases de interés e invariantes a todas sus posibles versiones (p. ej. cambios en posición, tamaño, orientación, intensidad de color, etc.). Un conjunto de propiedades de mala calidad produce un solapamiento de clases y por tanto una gran probabilidad de error en la clasificación. Como la anterior, esta etapa también es esencial para el éxito de cualquier sistema de percepción. Desafortunadamente, excepto en casos relativamente sencillos, no existen técnicas automáticas estándar que, aplicadas a la información original, obtengan resultados discriminantes e invariantes. En la mayoría de los casos el diseñador debe disponer de un conocimiento sobre el problema que le indique propiedades o atributos potencialmente útiles. Posteriormente estas características candidatas sí pueden ser evaluadas y seleccionadas de manera automática.

- Reconocimiento: en esta etapa se decide la categoría más probable (dentro de un conjunto preestablecido) a la que pertenece cada observación sensorial elemental caracterizada por un vector de propiedades. El reconocimiento tiene un planteamiento matemático bien definido y puede considerarse resuelto desde el punto de vista de la teoría de la decisión. Existen diferentes enfoques, entre los que destaca el probabilístico-estadístico. Además, cuando el conjunto de propiedades obtenidas en la etapa anterior es suficientemente discriminante, la complejidad de esta etapa se reduce sensiblemente. En caso contrario, un diseño correcto del reconocedor contribuirá, al menos, a disminuir la proporción de errores.

- Interpretación: los conceptos elementales obtenidos en la etapa anterior se organizan en una estructura espacio-temporal requerida por la aplicación. En algunos casos existen algoritmos eficientes, pero en general se basan en algún tipo de búsqueda heurística que trata de evitar la explosión combinatoria de interpretaciones alternativas. En problemas de percepción de bajo nivel esta etapa no suele ser necesaria.

Como vemos, existe una gran variedad en la naturaleza de cada etapa y en el tipo de herramientas matemáticas e informáticas utilizadas para resolverlas. A medida que la información avanza a través de la cadena de proceso las etapas tienden a estar cada vez mejor definidas y a depender menos de la aplicación concreta, excepto la adquisición inicial, que utiliza normalmente dispositivos estándar, y la interpretación, que produce el resultado final del sistema.

Esta descomposición en subtareas aparentemente más sencillas, que se tratan de resolver por separado, es una guía razonable para abordar el diseño de sistemas de reconocimiento. De hecho, la modularización es la manera habitual de atacar la complejidad. Pero no debemos olvidar que las etapas están íntimamente relacionadas unas con otras y no es fácil resolverlas independientemente. Recordemos además que, excepto en casos muy

simples, la información se mueve en los dos sentidos, ascendente y descendente. Una de las causas de la naturaleza computacionalmente intratable de la percepción es la dificultad de descomponerla en módulos bien definidos e independientes.

El éxito de un sistema de percepción artificial depende de la calidad y eficiencia que podamos conseguir en las diversas etapas, principalmente en aquellas que dependen de la aplicación concreta y no son fácilmente automatizables. El objetivo primordial es conseguir sistemas de percepción capaces de operar satisfactoriamente en ambientes cada vez menos controlados.

En la Figura 1.1 se muestran las etapas de un reconocedor de formas bidimensionales. En esta tesis nos centraremos en este tipo de sistemas, más concretamente, en las etapas de parametrización (es decir, supondremos que la figura está ya segmentada) y sobre todo en la de reconocimiento.

1.3. Aproximaciones al reconocimiento

Se pueden encontrar dos aproximaciones a este problema, motivadas por la diversidad de las tareas de reconocimiento que pueden abordarse: el reconocimiento estadístico de patrones (REP) [DH73] y el reconocimiento sintáctico de patrones (RSP) [Fu82] (también llamado estructural).

El REP consiste en representar cada patrón mediante un vector de números resultantes del muestreo y la cuantificación de las señales externas (vector de características) y cada clase por algún modelo obtenido a partir de uno o varios patrones prototipo. De este modo, un patrón no es más que un punto en el espacio de representación de los patrones, que es un espacio de dimensionalidad determinada por el número de variables consideradas. Esta representación hace que sea sencillo aplicar sobre los objetos análisis estadístico (de ahí su nombre). Dado que existe variabilidad en las medidas registradas, cada componente del vector es una variable aleatoria y cada uno de sus valores es una realización de esa variable aleatoria.

La aproximación estadística no considera la relación entre diferentes patrones: en ocasiones, patrones complejos pueden descomponerse recursivamente en patrones más simples hasta llegar a componentes básicos (de forma similar a cómo un texto se descompone en párrafos, frases, palabras y finalmente en letras). En el RSP la representación de los objetos se realiza habitualmente mediante cadenas o estructuras más complejas como grafos o árboles. Con esta aproximación, un patrón se describe en términos de sus elementos básicos (elementos terminales) y unas reglas sintácticas (gramática) que especifican cómo se pueden generar patrones válidos de una determinada clase. El problema se reduce entonces a responder si un determinado patrón pertenece al lenguaje generado por una gramática. Para el manejo de estas estructuras se pueden aplicar los resultados aportados por la teoría de lenguajes formales [AU73, HU79].

Hay que destacar que la frontera que separa ambas ramas es bastante difusa y que algunas técnicas que se aplican de manera más habitual en una rama pueden adaptarse a la otra. Incluso hay técnicas que hacen uso de ambas aproximaciones, téngase en cuenta el caso de los modelos ocultos de Markov [Rab89].

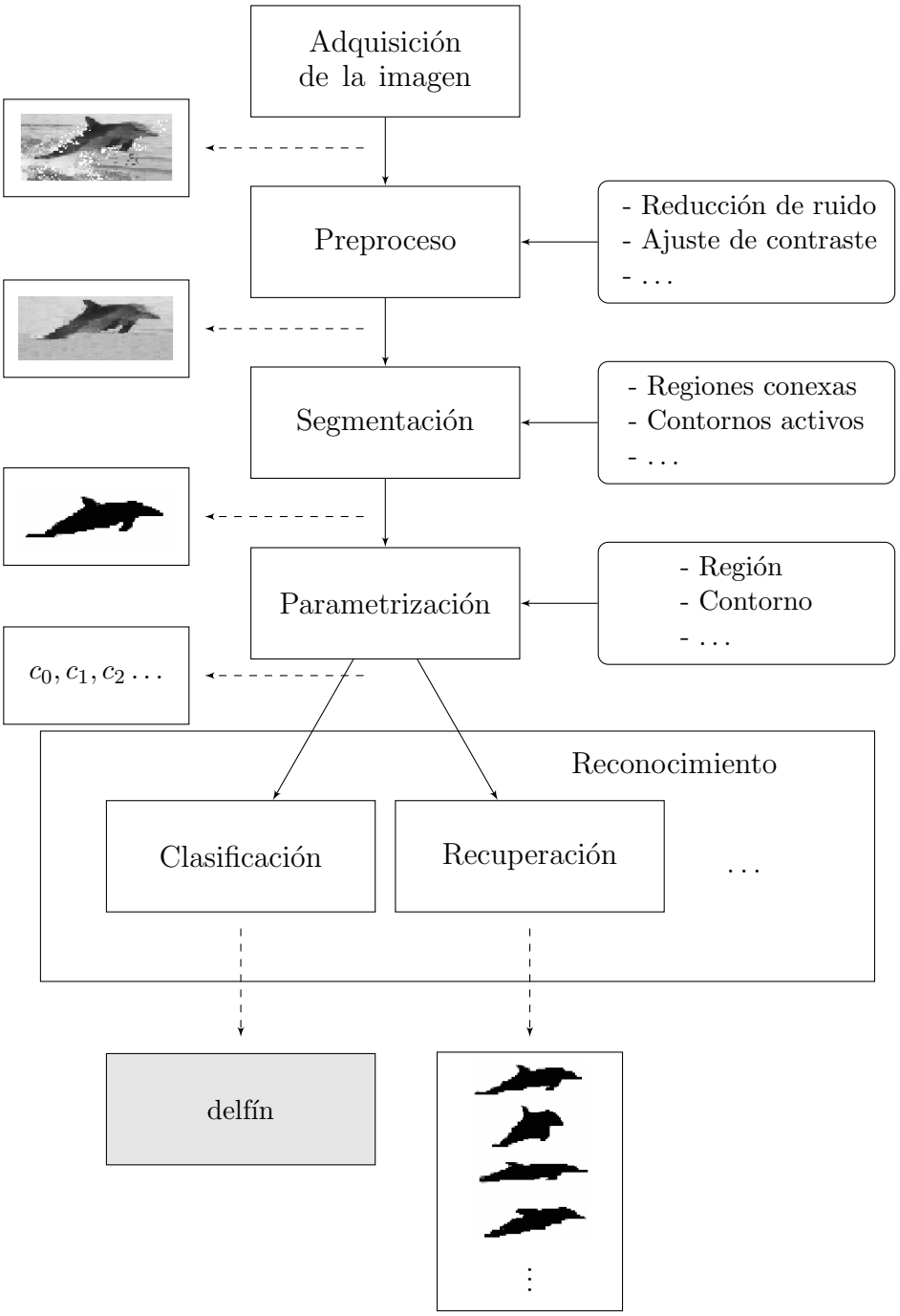


Figura 1.1: Etapas de un sistema reconocedor de formas bidimensionales.

Siguiendo tanto la aproximación de REP como la de RSP, el funcionamiento de un sistema de RP es básicamente el mismo. Existe una primera fase de entrenamiento, en la cual se le aportan al sistema los datos necesarios para que mediante un determinado proceso, aprenda la naturaleza de cada una de las categorías y/o el universo de posibles interpretaciones. Tras esta fase, se puede pasar a la utilización del sistema, donde se aporta un objeto desconocido y este, tras un proceso, responde a qué categoría o clase pertenece. Atendiendo a la forma del proceso que lleva a cabo el sistema de RP tanto en la fase de entrenamiento (cómo representa la naturaleza de las categorías) como en el uso (cómo utiliza la representación de las categorías para hacer el reconocimiento), se habla entonces de métodos de reconocimiento paramétricos y de métodos de reconocimiento no paramétricos.

En los métodos de reconocimiento paramétricos los modelos vienen descritos por valores asignados a características fundamentales (parámetros). Es decir, en la fase de entrenamiento se extraen a partir de los datos los parámetros de los modelos y en la fase de reconocimiento se utilizan estos modelos para verificar la pertenencia o no del objeto a la categoría representada por el modelo.

Los métodos de reconocimiento no paramétricos se caracterizan por la ausencia de un modelo basado en parámetros, ya que en este caso son los propios objetos (o prototipos) los que actúan como modelos. De esta manera, el reconocimiento de un objeto se realiza según el valor de la distancia entre este y los prototipos.

1.4. Problemas y aplicaciones en el reconocimiento de formas bidimensionales

El reconocimiento de formas es estudiado de diferentes maneras. Dadas dos representaciones de formas bidimensionales y una medida de disimilitud, podemos:

- calcular la disimilitud entre estas dos formas,
- para un umbral dado, decidir si la disimilitud es menor que este umbral,
- para un umbral dado, decidir si existe una transformación tal que la disimilitud entre la forma transformada y la otra es menor que este umbral,
- encontrar la transformación que minimiza la disimilitud entre la forma transformada y la otra.

Estos problemas juegan un papel importante en las siguientes categorías de aplicación:

- Clasificación: determinar si una forma dada, se acerca lo suficiente (por ejemplo, utilizando una medida de disimilitud) a formas que han sido previamente etiquetadas con la categoría o clase a la que pertenecen, para decidir si esta pertenece a la categoría.
- Agrupamiento: a partir de un conjunto de formas, determinar las clases o categorías de estas sin tener un conocimiento a priori.

- Búsqueda: para una imagen y una forma dadas determinar si la forma se encuentra en la imagen e indicar su situación.
- Recuperación: para todas las formas de una base de datos encontrar las que son más similares a una forma dada. Si la base de datos es muy grande suele ser inabordable calcular la similitud entre la forma dada y todas las que están en la base de datos. Los denominados métodos de indexación excluyen grandes partes de la base de datos antes de realizar la comparación.
- Transformación: convertir una forma de manera que se alinee con la otra del mejor modo. Es un problema de optimización y puede aplicarse a toda la forma o sólo a una parte.
- Aproximación: reconstruir la forma con pocos elementos (puntos, segmentos, etc.), para obtener otra similar a la original.

En esta tesis abordaremos las aplicaciones: clasificación y recuperación de formas. A lo largo de ella, cuando utilicemos los términos «reconocimiento» o «reconocimiento de formas», nos estaremos refiriendo a ambas aplicaciones. En la recuperación de formas utilizaremos técnicas no paramétricas y en la clasificación utilizaremos tanto no paramétricas, como paramétricas.

1.5. Cadenas cíclicas

A la hora de comparar dos formas bidimensionales nuestra representación de la forma y nuestra disimilitud deben tener en cuenta invarianzas a muchas distorsiones, incluyendo escalado, traslación, ruido, etc. La mayoría de estas distorsiones son relativamente fáciles de manejar, sin embargo, no importa qué representación utilicemos, se nos presenta un problema que entraña cierta dificultad: la elección del punto inicial con el que representamos el contorno como secuencia. Este problema es especialmente acusado cuando las formas pueden aparecer rotadas, pues ciertos heurísticos de selección del punto inicial se ven seriamente afectados por esta transformación. En la Figura 1.5 hay dos parametrizaciones (utilizando el contorno) de la misma forma bidimensional, pero con una rotación diferente. Aparecen también dos puntos iniciales que son los que seguramente produciría nuestro procedimiento de extracción del contorno (en este caso, el punto que está más arriba y a la izquierda). Se puede ver que el orden de los datos parametrizados del contorno cambiará drásticamente de acuerdo a la rotación y, por tanto, la comparación entre estos carecerá de sentido.

Para solucionar el problema de la invarianza al punto inicial existen tres enfoques: elección de una rotación de referencia, elección de características invariantes a la rotación y comparación de secuencias considerando todo posible punto inicial.

La idea de la elección de una rotación de referencia consiste en encontrar una rotación canónica para todas las formas y, a partir de esta, seleccionar el punto inicial con algún criterio definido. Sin embargo, es una técnica que se basa en heurísticos que suelen fallar en dominios no restringidos y un buen alineamiento en la rotación es vital para

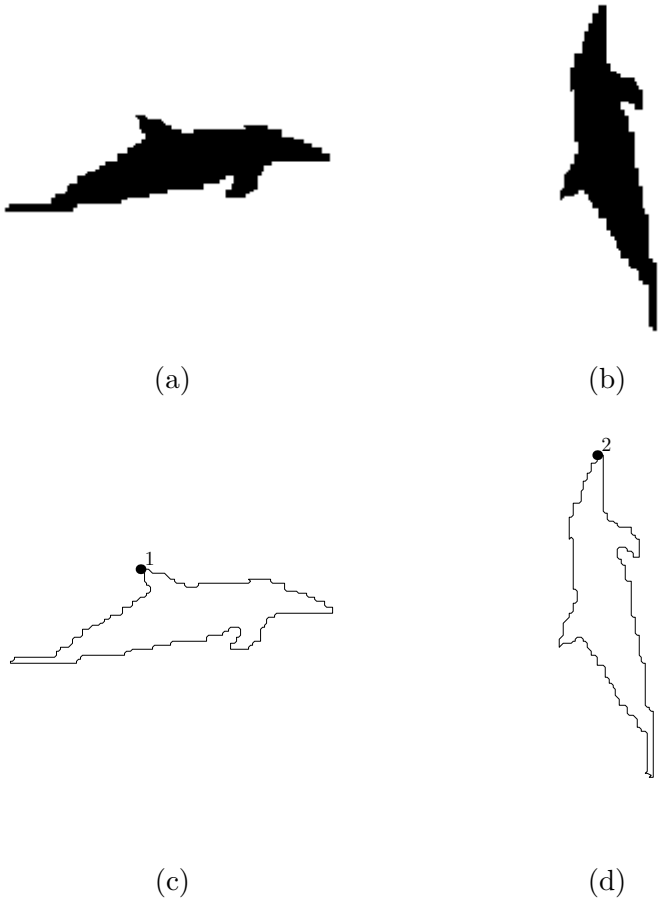


Figura 1.2: Ejemplo de parametrización con contornos con puntos iniciales diferentes. (a) Forma original. (b) Forma original que aparece rotada 90 grados. (c) y (d) Parametrizaciones del contorno correspondientes, con la elección de punto inicial como el que está más arriba y a la izquierda, etiquetados como «1» y «2», respectivamente.

conseguir resultados adecuados. Como segundo enfoque tenemos la utilización de vectores con características invariantes a la rotación del contorno. Existen multitud de estas características incluyendo proporción entre el perímetro y el área, circularidad, elongación, momentos, espectro de Fourier, etc. Pero en general y con la excepción del espectro de Fourier, estos métodos ofrecen una discriminación muy pobre sobre las formas que dificulta el reconocimiento de estas.

Los dos métodos anteriores no son satisfactorios en muchos dominios de aplicación. La verdadera invarianza al punto inicial se consigue midiendo distancias con todos los posibles puntos iniciales, es decir, utilizando el alineamiento por fuerza bruta. De aquí surge el concepto de cadena cíclica. Las cadenas cíclicas son cadenas de símbolos o valores que no tienen principio ni final, es decir, una cadena cíclica modela el conjunto de todos los posibles desplazamientos cíclicos de una cadena y medir una distancia entre dos cadenas cíclicas sería lo mismo que medir una distancia entre todos los posibles puntos iniciales de las dos cadenas.

Existe mucho trabajo en el dominio de las cadenas y las distancias de edición (véase el Capítulo 3) en cuanto a las cadenas cíclicas se refiere. Pero si nos encontramos con cadenas de valores continuos y pretendemos conseguir invarianza al ruido y otras deformaciones, estas técnicas no suelen dar buenos resultados. Otras técnicas suelen ser más adecuadas en estos casos, como el alineamiento temporal no lineal [SK83] o los modelos ocultos de Markov [Rab89] (introduciremos ambas técnicas en el Capítulo 4). Sin embargo, no existe demasiado trabajo que relacione estas técnicas con las cadenas cíclicas. Esta tesis pretende humildemente llenar un poco este vacío.

Primeramente, definimos el alineamiento temporal no lineal cíclico (ATNLC) y un método eficiente para su cálculo basado en la técnica algorítmica divide y vencerás. Se extiende trabajo similar en el campo de la edición de cadenas cíclicas, pero detectando y corrigiendo errores comunes en una adaptación naíf del algoritmo de Maes. Se incluyen además diversas extensiones, como otras ponderaciones en las producciones y la normalización de la distancia.

Proponemos también otras técnicas para acelerar este cálculo en tareas de reconocimiento. En primer lugar, si en nuestra tarea de reconocimiento tenemos categorías, podemos utilizar el siguiente heurístico. Seleccionar un buen punto de inicio de todas las cadenas de cada categoría en base a un buen representante de esta, utilizando el ATNLC, de manera que, a la hora de realizar el reconocimiento, en la mayoría de las ocasiones sólo habrá que utilizar el alineamiento temporal no lineal (no cíclico), reduciendo el coste considerablemente. En segundo lugar, tenemos la aceleración con una técnica que se basa en utilizar una cota inferior basada en una aproximación a la distancia cíclica. Por último, se aportan soluciones para utilizar AESA (*Approximating and Eliminating Search Algorithm*), aun no siendo el ATNLC una métrica, algo necesario para la utilización de este algoritmo. Además también introducimos un nuevo algoritmo que mejora el LAESA (*Linear* AESA), al que hemos llamado LAESAEA (LAESA *with Early Abandon*).

En el terreno de los modelos ocultos de Markov (MOM) proponemos la utilización de una topología lineal como la mejor opción para el reconocimiento de contornos. Aportamos extensiones a los algorimos de decodificación (algoritmo de Viterbi) y entrenamiento (algoritmos de Viterbi y Baum-Welch) para modelar las cadenas cíclicas. Por último, vemos

que el heurístico comentado para el ATNLC también lo podemos utilizar con los MOMs para acelerar el reconocimiento y el entrenamiento.

Hemos organizado la tesis de la siguiente manera:

- Capítulo 2. Comentamos el estado del arte en la parametrización y el reconocimiento de contornos.
- Capítulo 3. Comentamos el estado del arte de las cadenas cíclicas con las distancias de edición.
- Capítulo 4. Comentamos el estado del arte en el reconocimiento de contornos con el alineamiento temporal no lineal y los modelos ocultos de Markov.
- Capítulo 5. Presentamos un algoritmo eficiente para el cálculo del ATNLC.
- Capítulo 6. Presentamos diversas técnicas para la aceleración del ATNLC en tareas de reconocimiento. Óptimas y aproximadas.
- Capítulo 7. Presentamos técnicas para la utilización de las cadenas cíclicas con los modelos ocultos de Markov.
- Capítulo 8. Realizamos un estudio experimental de las diferentes propuestas de la tesis.
- Capítulo 9. Mencionamos las conclusiones, aportaciones de la tesis y futuras líneas de investigación.

Parte I

Estado del arte

Capítulo 2

Parametrización y reconocimiento de formas bidimensionales con contornos

Pirate: «*So we're taking all 15 of them?*»
Head Pirate: «*Well it wouldn't be nice to separate them from their friends.*»
Hayao Miyazaki, *Pòrco Rósso* (1998).

En este capítulo presentaremos nuestra visión general y jerárquica sobre la parametrización de formas bidimensionales y las metodologías que existen para su reconocimiento utilizando contornos. Comentaremos también de una manera genérica, ya que es el problema principal de esta tesis y se abordan con más extensión en posteriores capítulos, las diferentes propuestas de la literatura para solucionar el problema de la invarianza al punto inicial. Por último, mencionaremos los métodos que se suelen utilizar para la evaluación de los sistemas de reconocimiento.

El problema del análisis de formas bidimensionales ha sido abordado por muchos investigadores por lo que hay una gran cantidad de trabajo en la literatura. No ha sido fácil distribuir los métodos de una manera jerárquica. Es muy posible que en algunos casos, tanto métodos de parametrización como de reconocimiento tengan también cabida en otras posiciones de la jerarquía, o fuera de ella. Hemos considerado pues, que esa era la mejor posición o por lo menos la más común. El lector puede referirse a libros estándar de la literatura para más información sobre procesamiento digital de formas [GW92, Jäh01, Jai89] y reconocimiento de patrones [Bis06, DH73, DHS01]. Existen también artículos con buenas revisiones sobre el tema [Kin03, Lon98, VH01, ZL04].

2.1. Introducción

Parametrizar una forma bidimensional significa extraer las características de esta para describir cualitativa o cuantitativamente sus propiedades importantes. Este conjunto de características extraídas de una forma, a las que llamaremos «descriptor», pueden no ser suficientes para reconstruirla. En el caso de que sean suficientes a esta extracción se le

llama representación de la forma[1].

Los métodos para parametrizar formas pueden clasificarse como:

- Descriptores de región: aquellos que se basan en características de la región que ocupa la forma.
- Descriptores de contorno: aquellos que se basan en el contorno exterior de la forma (véase la Sección 2.2).

Del mismo modo para reconocer estas formas, utilizando sus descriptores, necesitaremos de un método de reconocimiento (véase la Sección 2.3):

- Métodos estadísticos no paramétricos: basados en distancias y en un conjunto de prototipos. Siendo estos los más utilizados en el reconocimiento de formas debido a su sencillez y buenos resultados.
- Médodos estadísticos paramétricos: basados en el entrenamiento de un modelo con un conjunto de prototipos.

La efectividad de un sistema reconocedor de formas dependerá, en gran medida, del descriptor y del método de reconocimiento de estos descriptores. Para este fin, existen algunos criterios que deben de ser considerados (en la Sección 2.5 veremos métodos para evaluar algunos de estos criterios, directa o indirectamente):

- Unicidad: una forma necesita ser representada de manera única, de otro modo, la forma reconocida puede no ser similar a la que se pretende identificar.
- Invarianza: es muy recomendable que el descriptor y el método utilizado para medir la similitud sean invariantes a transformaciones geométricas, ruido u otras deformaciones.
- Estabilidad: las pequeñas variaciones en la forma harán que tanto la representación como la medida se vean afectados de igual manera con pequeñas variaciones.
- Escalabilidad: la capacidad de discriminación del descriptor y del método de comparación no deben verse afectados (o al menos no demasiado) por la cantidad de formas que tengamos.
- Eficiencia: la obtención del descriptor y el método de comparación deben de ser computacionalmente eficientes. Idealmente permitirán trabajar en tiempo real.
- Compacidad: el descriptor debe ser compacto para un almacenamiento eficiente.

Debido a que la presente tesis está centrada en los métodos basados en contornos (más concretamente en la invarianza al punto de inicio de estos), no veremos en este capítulo los descriptores de región.

[1]Por simplicidad, en este documento, utilizaremos ambos términos para referirnos al mismo concepto.

2.2. Descriptores de contorno

Como su nombre indica estos descriptores sólo utilizan la información del contorno de la figura. Las propiedades del contorno son muy importantes para la percepción humana en cuanto a la búsqueda de similitud y reconocimiento de formas. La mayoría de autores que estudian los sistemas de percepción visual del ser humano están de acuerdo en lo significativos que son los puntos de mucha curvatura [Att54, Mar76]. Los experimentos psicológicos sugieren que las esquinas tienen un alto grado de contenido informativo y, al describir una figura, las esquinas corresponden a puntos de mucha curvatura. Por lo tanto, el contorno de la forma contiene más información que el interior de esta, en términos de percepción.

Muchas de las parametrizaciones descritas a continuación pueden utilizarse también combinadas. Por ejemplo, los métodos de selección de puntos (véanse las Secciones 2.2.4, 2.2.5 y 2.2.7) nos ofrecen una descripción de la forma con coordenadas que pueden utilizarse directamente con algunos métodos de reconocimiento, pero es muy común también utilizarlos para hacer una selección de puntos previa, cuando el número de puntos del contorno es muy grande, para evitar una parametrización o reconocimiento muy costosos [BMP02, AO04, ARKF07].

2.2.1. Descriptores simples

Existe un gran número de descriptores basados en un enfoque heurístico y/o topológico en los que la forma se describe con un escalar. Estos no permiten la reconstrucción de esta a partir de ellos, pero pueden ofrecer buenos resultados describiendo formas simples en entornos muy restringidos. Por otro lado, también nos pueden servir como un primer filtro muy rápido para descartar formas. Algunos de estos descriptores son los siguientes:

- Perímetro: longitud del contorno de la forma.
- Rectangularidad: proporción entre el área y el área de la caja circundante (*bounding box*) de la forma. La caja circundante es el rectángulo de menor área que contiene a la forma.
- Circularidad: proporción entre el área y un círculo con el mismo perímetro que la forma.
- Elongación: proporción entre la longitud y la anchura de la caja circundante de la forma.
- Orientación del eje mayor: el ángulo entre el eje mayor y el eje x. Siendo el eje mayor de una forma la línea recta que une los dos puntos más alejados del contorno.
- Excentricidad: proporción entre el eje mayor y el menor. Siendo el eje menor la línea perpendicular al eje mayor y la longitud de estos ejes la parte del eje que queda dentro de la forma.

2.2.2. Funciones unidimensionales

El contorno de una forma bidimensional, definido por una cadena de puntos, puede ser representado utilizando una función real o compleja de una dimensión con los métodos que se describen a continuación.

2.2.2.1. Funciones paramétricas

Considerando que el contorno de la figura F está en el espacio euclídeo. Todo punto $p_i(x(i), y(i))$ del contorno puede ser identificado con el valor i, con $0 \leq i < m$ (donde m es el número de puntos del contorno de la forma o su perímetro, dependiendo de estar en un espacio discreto o continuo), en el sentido contrario a las agujas del reloj[2]. Un punto que se mueve a lo largo del contorno genera una función que puede ser representada de manera paramétrica como $c(i) = (x(i), y(i))$.

Todo punto p_i puede expresarse en coordenadas polares $(d(i), \theta(i))$. Un punto que se mueva a la largo del contorno genera una función que puede ser representada de manera paramétrica como $\tau(i) = (d(i), \theta(i))$.

Finalmente, si consideramos que la figura está en el plano complejo. Todo punto p_i tiene sus coordenadas complejas $z(i) = x(i) + jy(i)$. Un punto que se mueve a lo largo del contorno genera una función compleja $z(i)$.

Nótese que las tres funciones de contorno son equivalentes. Es decir, cuando conocemos una de las variantes se pueden calcular fácilmente las demás:

$$z(i) = x(i) + jy(i)$$

$$c(i) = (Re(z(i)), Im(z(i)))$$

$$\tau(i) = (|z(i)|, arg(z(i)))$$

$$c(i) = (|d(i)| \cos(\theta(i)), |d(i)| \operatorname{sen}(\theta(i)))$$

En la Figura 2.1 se muestran ejemplos de una forma descrita de los tres modos. Estas funciones no son invariantes a la traslación de la forma, no son invariantes al escalado, y dependen de la orientación de F y del punto de inicio del contorno.

En algunas ocasiones es útil normalizar estas funciones con respecto al perímetro, con lo que el perímetro queda definido entre $[0, 2\pi]$ (también es habitual una normalización entre $[0, 1]$). El perímetro es normalizado definiendo $t = 2\pi i/m$ y las funciones son normalizadas como $c^*(t) = c(tm/(2\pi))$, $\tau^*(t) = \tau(tm/(2\pi))$ y $z^*(t) = z(tm/(2\pi))$.

[2]En este documento, todos las cadenas que describan un contorno seguirán este criterio a menos que se indique lo contrario.

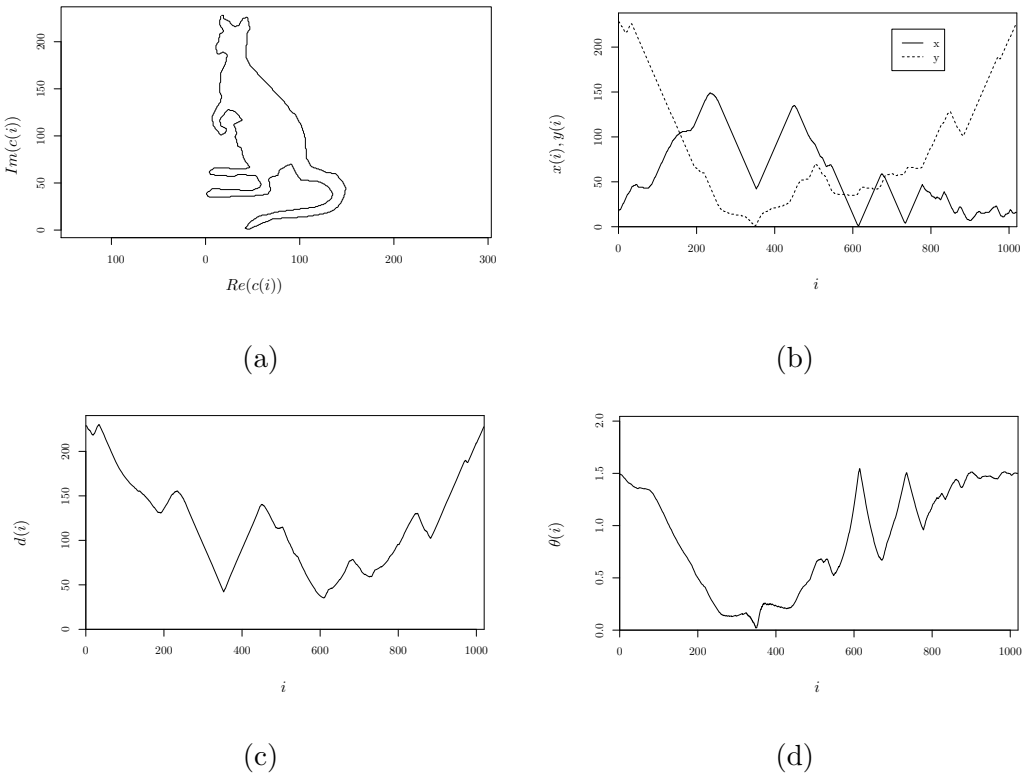


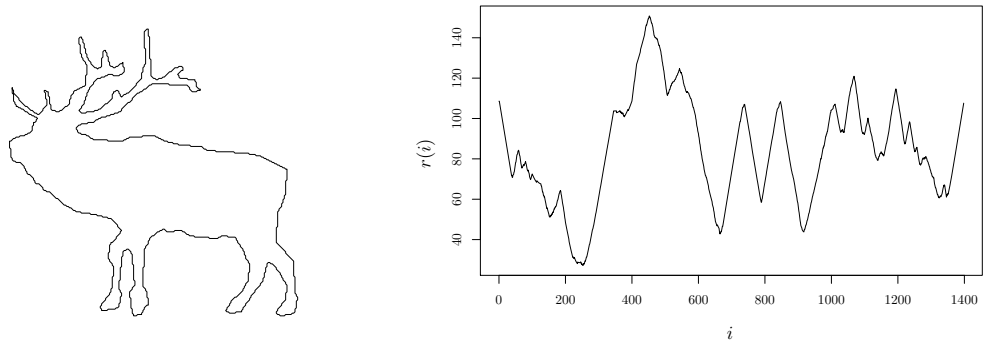
Figura 2.1: (a) Función compleja $z(i)$. (b) Función paramétrica $c(i) = (x(i), y(i))$. (c, d) Función paramétrica $\tau(i) = (d(i), \theta(i))$.

2.2.2.2. Radio-vector

Es una función que se utiliza frecuentemente [dBDLP98, Lon98, ZL02]. Si queremos obtenerla, en primer lugar, tendremos que escoger un punto de referencia O. Este suele ser el centro de masas $O = (\bar{x}, \bar{y})$ (a este caso se le conoce como distancia al centroide), pero también puede ser otro punto físicamente importante. A continuación, se calcula la función utilizando:

$$r(i) = ([x(i) - \bar{x}]^2 + [y(i) - \bar{y}]^2)^{1/2}.$$

La función $r(i)$ es invariante a la traslación. La rotación de la forma produce un desplazamiento cíclico y el escalado cambia $r(i)$ linealmente.



(a) (b)

Figura 2.2: Ejemplo de la función radio-vector. (a) Contorno de la figura. (b) Su correspondiente función radio-vector.

2.2.2.3. Ángulo de la tangente

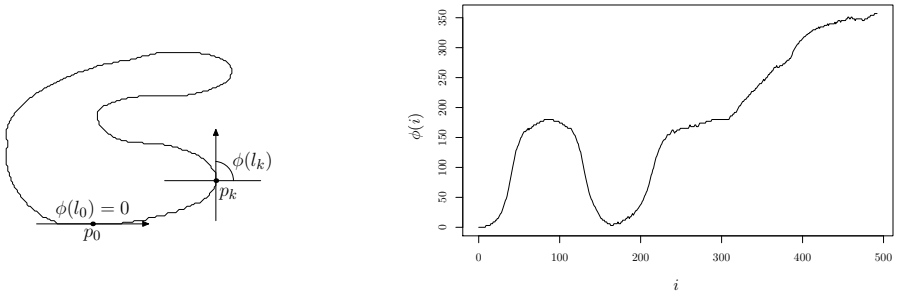
A menudo se utiliza como descriptor el ángulo de la tangente a lo largo del contorno [AcH$^+$91].

Para su obtención, se sitúa un puntero en p_0 siendo el ángulo de la tangente en ese punto igual a 0. Si el puntero se mueve a lo largo del contorno, cambia su dirección de manera que siempre está en la dirección de la tangente, donde su orientación es dada por la dirección del movimiento. El ángulo dado por la dirección del puntero en p_i viene dado por

$$\phi(i, k) = \arctan \frac{y(i) - y(i-k)}{x(i) - y(i-k)},$$

donde k es el tamaño de ventana. Un ejemplo puede verse en la Figura 2.3.

Algunas veces es útil normalizar la función $\phi(i)$ definiéndola entre $[0, 2\pi]$. Para realizarlo seguiremos el siguiente procedimiento. El perímetro es normalizado definiendo $t = 2\pi i/m$ y $\phi(i)$ es normalizado como $\phi^*(t) = \phi(mt(2\pi)^{-1}) + t$ donde $0 \leq t \leq 2\pi$ y $\phi^*(0) = \phi^*(2\pi) = 0$.



(a) (b)

Figura 2.3: (a, b) Definición de la función ángulo de la tangente $\phi(i)$ de la forma.

La función $\phi^*(t)$ es invariante a la traslación y cambios de tamaño en la figura. No depende de la orientación de la figura y es también una función periódica, con periodo 2π.

2.2.2.4. Curvatura

Se pueden dar tres definiciones diferentes de la curvatura [JdFC95, LS90, MM92]:

- curvatura continua basada en la orientación,

$$\kappa(i) = \phi'(i); \tag{2.1}$$

- curvatura continua basada en el camino,

$$\kappa(i) = \frac{x'(i)y''(i) - x''(i)y'(i)}{(x'(i)^2 + y'(i)^2)^{3/2}}; \tag{2.2}$$

- curvatura continua basada en el círculo asociado,

$$\kappa(i) = \begin{cases} +\frac{1}{\rho(i)} & \text{si el contorno es localmente convexo,} \\ -\frac{1}{\rho(i)} & \text{si el contorno es localmente cóncavo,} \end{cases}$$

siendo $\rho(i)$ el radio de la circunferencia que más se asemeja a la curva en el punto i.

Las tres definiciones son equivalentes en el caso continuo pero no en el discreto. El uso de segundas derivadas produce que el cálculo de la curvatura sea impracticable con curvas discretas. De modo que se debe de hacer algún tipo de suavizado sobre las curvas antes de aplicar (2.2), como por ejemplo, un suavizado gaussiano [MM86]. En la Figura 2.4 podemos ver un ejemplo.

La función curvatura puede ser calculada como la derivada de la función ángulo tangente (2.1). Para prevenir el ruido causado por las fluctuaciones de los contornos se puede aplicar también un suavizado, simplemente utilizando una ventana k adecuada al calcular

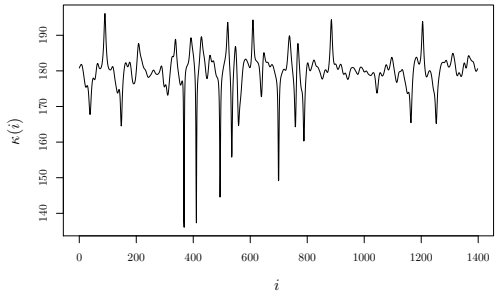


Figura 2.4: Función curvatura $\kappa(i)$ de la Figura 2.2a, después de aplicarle un suavizado gaussiano.

la función ángulo tangente (Sección 2.2.2.3). Podemos definir así la k-curvatura, en un punto del contorno, como

$$\kappa(i,k) = \phi(i,k) - \phi(i+k,k).$$

La curvatura tiene las mismas propiedades a las mencionadas para la función de ángulo de la tangente. Algunas características geométricas del contorno están relacionadas con la curvatura. Por ejemplo, las esquinas están localizadas en partes del contorno donde la curvatura es muy alta.

Otro atributo asociado a la curvatura es la energía (*bending energy*):

$$E = \frac{1}{m}\int_0^m |\kappa(i)|^2 di,$$

donde m es el número de puntos o perímetro del contorno.

2.2.3. Códigos de Freeman o de cadena

Los códigos de cadena fueron introducidos por Freeman [Fre74]. En este enfoque, el contorno es representado por una cadena de vectores de longitud uno y un conjunto de posibles direcciones. Escogiendo un punto inicial, el código de cadena puede ser generado utilizando cuatro u ocho direcciones como puede verse en la Figura 2.5. Un código de cadena con N direcciones es también posible [SF81].

Aunque la derivada del código de cadena [GW92] (en el caso de la Figura 2.5c, el código de cadena es 0101012223333 y su derivada sería 113131100100) pueda parecer invariante a las rotaciones, esto no es cierto ya que sólo es invariante a rotaciones múltiplo de $\pi/2$ radianes, y en general, el código de contorno depende de la orientación de la rejilla (nos referimos a una rejilla montada por nosotros o a la autoimpuesta por los píxeles). Una manera de evitar este problema es utilizando una rejilla que esté orientada con el eje mayor (definido en la Sección 2.2.1). Para resolver el problema del punto inicial se puede utilizar lo que es conocido como *shape number*, definido como el número menor que podemos formar con un desplazamiento cíclico de la representación de la derivada. En el caso de la Figura 2.5(c), el *shape number* sería: 001001313101. A pesar de todas estas mejoras, estos métodos son muy sensibles al ruido.

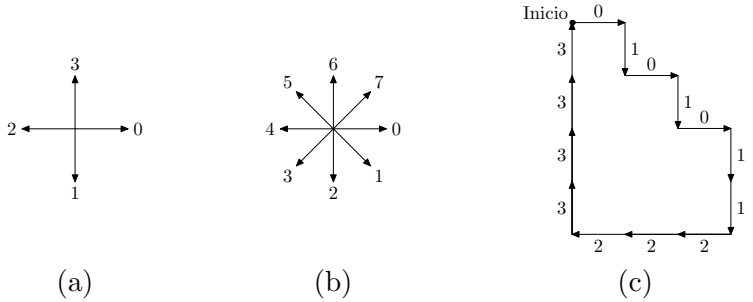


Figura 2.5: (a) Esquema de numeración de código de cadena para 4 direcciones. (b) Para 8 direcciones. (c) Código de cadena de 4 direcciones de una figura simple.

2.2.4. Aproximación equidistante

La aproximación equidistante se obtiene simplemente dividiendo el número de puntos por la cantidad de puntos que queremos obtener en la selección, para determinar el intervalo y con este intervalo realizar la selección.

Aunque puede utilizarse con algunos métodos de reconocimiento, es más frecuente utilizar esta aproximación como paso previo a otras parametrizaciones, con el objetivo de reducir y normalizar el número de puntos para todas las formas [BMP02, AO04, ARKF07].

2.2.5. Aproximación poligonal

La aproximación poligonal es uno de los principales métodos para representar contornos [Lon98]. La idea principal consiste en eliminar puntos del contorno seleccionando puntos de corte basándose en algún criterio de aproximación, como el error mínimo, el polígono con menor perímetro o el área de polígono máxima, entre otros.

El problema puede ser formulado de dos maneras:

- Problema de mínima distorsión: dado un contorno de n puntos, aproximarlo con un polígono de m de vértices de manera que se minimice el error de la aproximación.
- Problema de mínimo número de puntos: dado un contorno de n puntos y dado un error ϵ, aproximar este contorno con un polígono minimizando el número de vértices m.

En la literatura se conocen también como *min*-ϵ y *min*-# [KF07], respectivamente.

Estos problemas pueden ser resueltos con algoritmos de programación dinámica [PV94, CC96]. Existen también soluciones basadas en el algoritmo de búsqueda A* [Sal01].

Estos algoritmos tienen un coste elevado y es por esto que existen métodos basados en heurísticos para acelerar el proceso [HL02, KF07] que obtienen resultados subóptimos.

2.2.6. Aproximación con splines

La función paramétrica del contorno $c(i)$ puede dividirse en segmentos finitos. Cada uno de estos segmentos de curva puede ser aproximado con un polinomio [FvD96].

Las B-splines uniformes aproximan series de $m+1$ puntos de control ($p_0, \ldots, p_m, m \geq 3$) con una curva de $m-2$ segmentos polinómicos cúbicos $Q_3, \ldots, Q_m$. De este modo, para $m = 3$, Q_3 se define con $p_0, \ldots, p_3$, Q_4 se define con $p_0, \ldots, p_4$, y así sucesivamente:

$$Q_k(i - i_k) = \frac{(1-i)^3}{6} p_{k-3} + \frac{3i^3 - 6i^2 + 4}{6} p_{k-2} + \frac{-3i^3 + 3i^2 + 3i + 1}{6} p_{k-1} + \frac{i^3}{6} p_k.$$

Q_k y Q_{k+1} son continuos en i_k. Los puntos de unión i_k entre Q_k y Q_{k+1} se llaman *knots*. En el caso de las B-splines uniformes, los *knots* son equidistantes.

Por otro lado, existen las β-splines que requieren dos parámetros adicionales, β_1 y β_2, que proporcionan mayor control sobre la curva. Otras curvas cúbicas paramétricas que son utilizadas para aproximar contornos son las curvas de Bezier, las de Hermite, las B-splines no uniformes y los polinomios cúbicos racionales [FvD96].

2.2.7. Transformada de Fourier

La transformada de Fourier es uno de los métodos más utilizados en el análisis de formas [DH73, GW92, Jai89, PF77, ZL05].

Una función periódica continua y diferenciable $f(x)$ definida en $[0, 2\pi]$ (que puede ser cualquiera de las funciones unidimensionales mencionadas en secciones anteriores), puede ser aproximada con series de Fourier (en su forma compleja):

$$f(x) = \sum_{k=-\infty}^{\infty} c_k e^{jkx}, \tag{2.3}$$

donde

$$c_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) e^{-jkx}\, dx, \tag{2.4}$$

$$c_k = \frac{1}{2}(a_k - j b_k),$$

es decir, la transformada de Fourier (2.4) y la transformada inversa de Fourier (2.3) para volver a la señal original. En nuestro caso, utilizaremos la transformada discreta de Fourier (DFT) y teniendo en cuenta que la función $f(i)$ está definida entre $[0, m]$,

$$c_k = \frac{1}{m} \sum_{i=0}^{L} f(i) e^{-jki2\pi/m},$$

siendo la transformada inversa,

$$f(i) = \sum_{k=-m/2}^{m/2} c_k e^{jki2\pi/m}.$$

La función c_k está comprendida entre $[-m/2, m/2]$. Cada término de las series de Fourier se llama armónico y $A_k = \sqrt{a_k^2 + b_k^2}$ es la potencia del armónico. La secuencia formada por todos los A_k se llama espectro de potencia (véase la Figura 2.6).

En ocasiones es suficiente considerar sólo las frecuencias más bajas, es decir, quedarnos sólo con los primeros A_k/A_0. Con ello conseguimos dos cosas, eliminar ruido o detalles de alta frecuencia que no nos interesan y disminuir el número de componentes. Para ello, primero aplicamos la transformada de Fourier, seleccionando un número determinado de componentes de la transformada (que equivale al número de puntos que queremos para representar la función) y finalmente se aplica la transformada inversa. El algoritmo FFT (*fast Fourier transform*) realiza esto en tiempo $O(m \log m)$. En la Figura 2.7 puede verse el resultado: una función radio-vector aparece suavizada, con muchos menos puntos y conservando el parecido con la señal original.

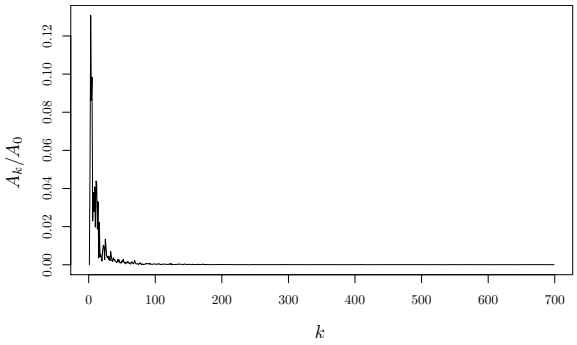


Figura 2.6: (a) Espectro de potencia normalizado A_k/A_0 para la función radio-vector mostrada en la Figura 2.2b. Sólo vemos una de las partes, la que se encuentra entre $[0, m/2]$.

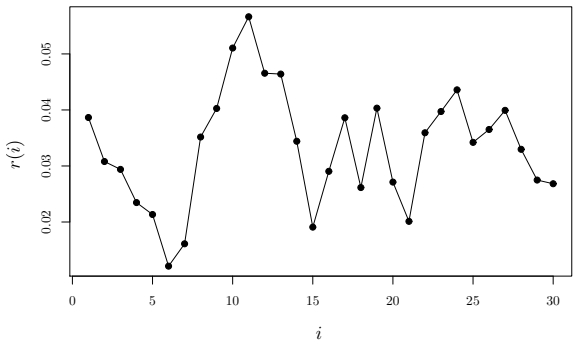
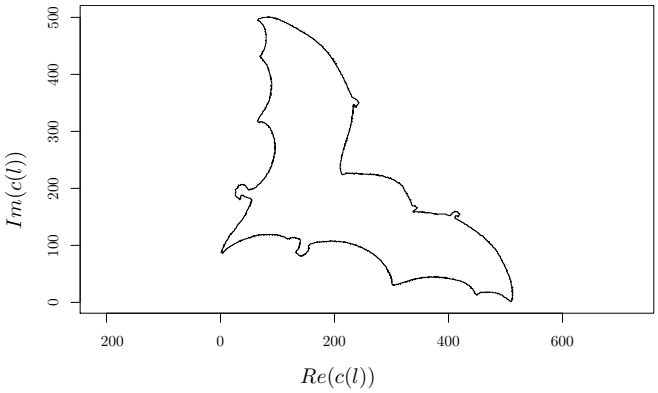


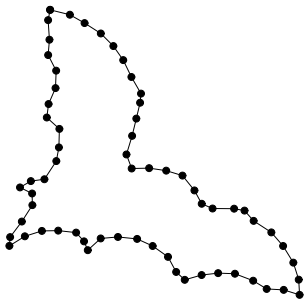
Figura 2.7: Función radio-vector de la Figura 2.2 después de pasarle el filtro paso bajo, seleccionando las 30 primeras componentes y normalizando con respecto a c_0.

Otra forma de tratar las formas bidimensionales con la DFT es el uso de los descriptores de Fourier (DF). Estos se consiguen aplicando la DFT sobre la representación del contorno con complejos: $z(i) = x(i) + jy(i)$, tal como puede verse en la Figura 2.8a. Del mismo modo que se ha explicado en párrafos anteriores también podemos realizar una selección de

frecuencias para reducir el número de puntos en la forma bidimensional. En la Figura 2.8b podemos ver un ejemplo donde hemos pasado de 3058 puntos a 64.



(a)



(b)

Figura 2.8: (a) Representación de la figura original en el plano complejo. (b) Selección de 64 puntos utilizando Fourier.

Con los DF, en el espacio frecuencial, podemos realizar operaciones que se convierten en transformaciones geométricas en el dominio espacial [BCP05]. Si la figura se ve afectada por una traslación el único coeficiente de Fourier que se verá afectado es el correspondiente a $k = 0$. Un cambio de escala del contorno de la figura por un factor α produce un escalado en los coeficientes, $c'_k = \alpha c_k$ (donde c_k y c'_k son los coeficientes de Fourier del contorno original y el escalado, respectivamente). La rotación del contorno por un ángulo θ_0 causa un desfase de θ_0, $c_k = c_k e^{j\theta_0}$. Cambiando el punto de inicio del contorno resulta en una modulación de los coeficientes, $c'_k = c_k e^{jki_0 2\pi/m}$ donde i_0 es el nuevo punto de inicio.

2.2.8. Espacio de escala de curvatura

El espacio de escala de curvatura (en inglés, *curvature scale space*, CSS) propuesto por Mokhtarian y Mackworth [MM86, MM92] es utilizado por el estándar MPEG-7 [Bob01, MB03] para representar y comparar contornos. El método parte de una representación paramétrica del contorno:

$$\Gamma = \{(x(i), y(i))|i \in [0, 1]\}.$$

Entonces, cada curva 1-D es convolucionada con diferentes gaussianas incrementando el valor de la desviación estándar σ (nivel de escala), con lo que se suaviza gradualmente el contorno para diferentes niveles de escala:

$$X(i, \sigma) = x(i) * g(i, \sigma), \quad Y(i, \sigma) = x(i) * g(i, \sigma).$$

Para cada escala, la curvatura de cada punto del contorno es obtenida con

$$\kappa(i, \sigma) = \frac{x'(i, \sigma)y''(i, \sigma) - x''(i, \sigma)y'(i, \sigma)}{(x'(i, \sigma)^2 + y'(i, \sigma)^2)^{3/2}}. \tag{2.5}$$

Igualando (2.5) a cero, encontramos los puntos de inflexión que corresponden a cada escala. Con estos puntos podemos dibujar una imagen, llamada imagen CSS (véase la Figura 2.9). Esta imagen nos muestra los puntos finales de los segmentos cóncavos a lo largo del contorno (el eje horizontal) para cada escala (el eje vertical). A medida que la escala se incrementa, el efecto de suavizado es mayor y el número de puntos de inflexión decrece hasta que el contorno es totalmente convexo.

Para comparar dos contornos se utilizan los puntos máximos [AMK99] (véase la Figura 2.9(b)). Los contornos de la imagen CSS que están por debajo de un umbral no se tienen en cuenta ya que se les considera producto del ruido.

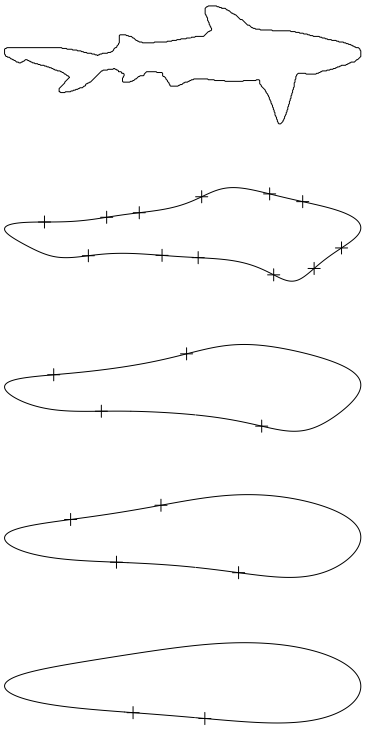
Esta representación es invariante a la escala, rotación y traslación. El método exhibe también cierta robustez ante transformaciones afines [AM01]. Sus principales limitaciones incluyen el que solo representa los segmentos cóncavos y que no es capaz de discriminar concavidades superficiales de las profundas. Por ejemplo, las figuras totalmente convexas como los cuadrados y triángulos tienen la misma imagen CSS, el vacío. Otra desventaja es el requerimiento de tener que calcular, en algunos casos, un número excesivo de escalas (más de 300).

2.2.8.1. Multi-scale convexity concavity

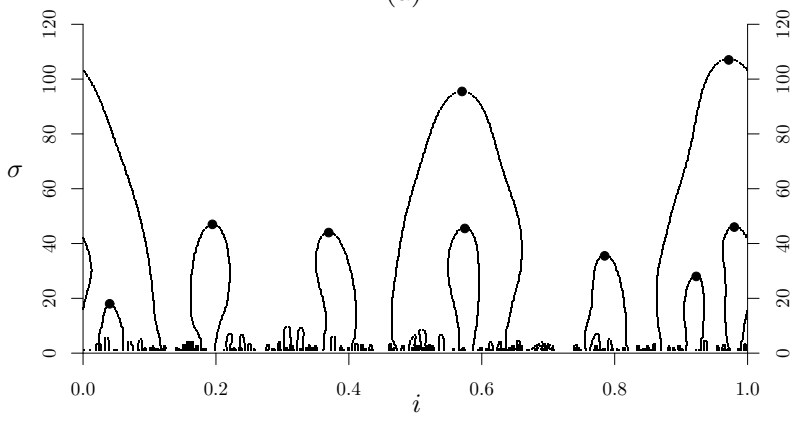
En [AO04], Adamek *et al.* proponen otro método multi-escala parecido al espacio de escala de curvatura (descrito en la Sección 2.2.8), pero utilizando una nueva medida para la curvatura basada en el desplazamiento relativo de los puntos del contorno con respecto a la posición en la escala de suavizado anterior. La idea se basa en la observación de que cuando se le aplica un suavizado a un contorno, los puntos convexos y cóncavos se mueven hacia dentro y hacia fuera, respectivamente. La cantidad de desplazamientos refleja el grado de curvatura. De este modo, cada punto está representado por un vector de desplazamientos.

2.2.9. Wavelets

Los *wavelets* han sido utilizados en multitud de aplicaciones. La razón de esto es porque es una poderosa herramienta matemática que nos proporciona localizaciones tiempo-frecuencia de señales y una representación jerárquica de estas.



(a)



(b)

Figura 2.9: (a) Figura original y suavizados (de arriba a abajo) para valores de σ, 40, 60, 80 y 100. Los puntos de inflexión están marcados con el símbolo «+»). (b) Imagen CSS. Los puntos máximos están marcados con el símbolo «•»).

La señal es pasada por filtros pasa-bajos y pasa-altos y de esta manera se separan las porciones de la señal de alta frecuencia de aquellas de baja frecuencia. Las componentes de baja frecuencia son las que otorgan a la señal la mayor parte de la información, las de alta se encargan de incorporar características más particulares. A partir de la señal S, con los filtros pasa-bajos se obtiene la señal A (aproximaciones) y a la salida de los pasa-altos la señal D (detalles). Se puede iterar el proceso de filtrado hasta llegar al nivel de precisión que se desee, obteniéndose un árbol de descomposición (véase la Figura 2.10).

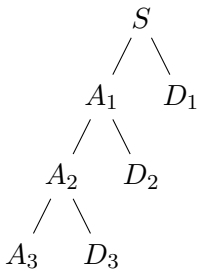


Figura 2.10: Árbol de descomposición wavelet en tres niveles.

Los coeficientes de la transformada *wavelet* han sido utilizados en [CK96, KBKNP96], como descriptores para el reconocimiento de formas. Otro enfoque se realiza en [AW99, TB95, KB02], donde los *wavelets* les sirven para obtener funciones invariantes a transformaciones afines. Por último, en [ERKA04, ERAK$^+$05], se propone un descriptor de formas multiescalar basado en *wavelets*, de donde se sacan momentos invariantes para cada nivel. Un ejemplo de esta descomposición con *wavelets* para una forma se puede ver en la Figura 2.11, en la que se obtienen 3 niveles. Quedando las aproximaciones a la izquierda y los detalles a la derecha.

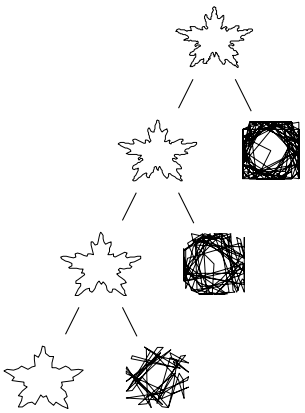


Figura 2.11: Ejemplo de descomposición wavelet en tres niveles de un contorno.

2.2.10. Representaciones estructurales

La representación de las formas se realiza aquí mediante cadenas, grafos o árboles [Fu82, GT78], descomponiendo el objeto en primitivas simples. El objetivo es conseguir una descripción recursiva o jerárquica a partir de estas primitivas.

En la Figura 2.12 se puede ver un ejemplo simple, donde el cromosoma quedaría descrito por la cadena «*babcbabdbabcbabd*». La categoría de estos cromosomas podría representarse con la siguiente gramática:

$$G = (\{S, A, B\}, \{a, b, c, d\}, S, P), \quad P = \{S \to AA, A \to BcBd, B \to bBb, B \to a\}.$$

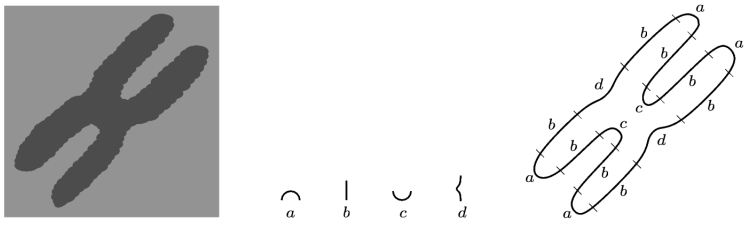


Figura 2.12: Descripción de un cromosoma con los elementos terminales: a, b, c y d.

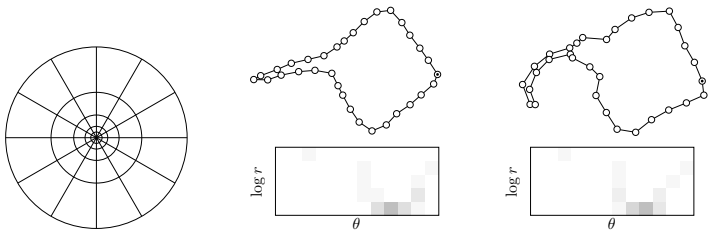
La principal ventaja de utilizar estos métodos radica en poder usar las técnicas de comparación sintácticas (véase la Sección 2.3.5). Ahora bien, una de los problemas en la descomposición de formas es el problema de la definición de las componentes. El resultado de la descomposición no siempre corresponde a la idea intuitiva humana de la representación de la forma. Además, los resultados no son siempre únicos. La descomposición de formas similares puede obtener árboles muy diferentes. En muchos casos, la derivación de la descomposición es costosa con una complejidad computacional relativamente alta comparada con otros métodos.

2.2.11. Puntos con información global

En esta sección agrupamos unos descriptores de contorno que como característica común tienen para cada punto (del contorno de la figura) un vector con información relativa de este con respecto a los demás puntos, es decir, información global. Por tanto, las cadenas obtenidas se componen típicamente de elementos de varias dimensiones.

2.2.11.1. Shape contexts

En [BMP02], Belongie *et al.* presentaron los *shape contexts*, obteniendo buenos resultados en clasificación y recuperación de formas. En su aproximación la forma se describe como una distribución espacial de puntos. Dado un conjunto de puntos x que describen la



(a) (b) (c)

Figura 2.13: (a) Diagrama del espacio log-polar para la obtención de los histogramas. Cinco *bins* para $\log(r)$ y 12 para θ. (b) y (c) Histogramas correspondientes de las formas en los puntos marcados.

forma, el *shape context* en el punto x_i se define como el histograma h_i de las coordenadas relativas de los $m-1$ puntos restantes[3]:

$$h_i(s) = \#\{x_j : j \neq i, x_j - x_i \in bin(s)\},$$

donde los *bins* son unas divisiones uniformes del espacio log-polar centrado en el punto x_i (véase la Figura 2.13a). En las Figuras 2.13b y 2.13c vemos como dos formas parecidas tienen histogramas similares en puntos similares.

Esta descripción del contorno como un conjunto de histogramas es invariante a la traslación. Es posible también conseguir invarianza al escalado normalizando las distancias radiales por la distancia media entre todos los pares de puntos de la forma. La invarianza a la rotación se consigue rotando el espacio log-polar con respecto a la tangente del contorno.

2.2.11.2. Beam angle statistics

El principal problema de la k-curvatura (Sección 2.2.2.4) es la selección de un valor para k que discrimine lo suficiente entre las formas a reconocer. Existen métodos para seleccionar este valor [AD97, UBS02], no obstante, en algunas ocasiones un sólo valor de k no es suficiente para conseguir esta discriminación.

Para resolver este problema el método presentado en [AYV03] utiliza todos los valores de k, y trata, para cada punto, estas curvaturas como una variable aleatoria, de manera que el descriptor está formado por momentos de estas variables. Es decir, en cada punto se obtienen todas las k-curvaturas posibles y se calculan los momentos centrales de estas. Los momentos de orden 1, 2 y 3 son los siguientes:

$$\Gamma^1(i) = \frac{1}{\frac{N}{2}-1}\sum_{k=1}^{\frac{N}{2}} \kappa(i,k),$$

$$\Gamma^2(i) = \sqrt{\frac{1}{\frac{N}{2}-1}\sum_{k=1}^{\frac{N}{2}} |\kappa(i,k) - \Gamma(i)|^2},$$

[3]El carácter «#» significa el número o cantidad.

$$\Gamma^3(i) = \sqrt[3]{\frac{1}{\frac{N}{2}-1}\sum_{k=1}^{\frac{N}{2}} |\kappa(i,k) - \Gamma(i)|^3}.$$

Los momentos describen el comportamiento estadístico de las k-curvaturas en los puntos. Cada punto es entonces representado por un vector cuyas componentes son los momentos.

2.2.11.3. Área del triángulo

La representación del área del triángulo (*triangle area representation*, TAR), se propone en [ARKF07].

Estos triángulos se forman para cada punto i, con la terna (x_{i-t_s}, y_{i-t_s}), (x_i, y_i), y (x_{i+t_s}, y_{i+t_s}), donde t_s es el incremento del índice y va de 1 a $T_s = \frac{m}{2} - 1$, siendo m el número de puntos del contorno. Por ejemplo, en el caso de que tuviéramos 100 puntos en el contorno, se calcularían 49 áreas de triángulos para cada punto. En la Figura 2.14 podemos ver ejemplos de estos triángulos.

El área del triángulo se puede calcular con

$$TAR(i, t_s) = \frac{1}{2}\begin{vmatrix} x_{i-t_s} & y_{i-t_s} & 1 \\ x_i & y_i & 1 \\ x_{i+t_s} & y_{i+t_s} & 1 \end{vmatrix}.$$

Los valores positivos, negativos y ceros corresponden a partes convexas, cóncavas y rectas, respectivamente.

Cada punto es entonces representado por un vector de valores formado por todas las áreas del triángulo para todos los posibles t_s.

2.3. Reconocimiento

Como se vio en la Sección 2.1, podemos diferenciar dos métodos de reconocimiento los paramétricos y los no paramétricos.

En los métodos de reconocimiento paramétricos los modelos vienen definidos por una serie de parámetros que hay que aprender. En la fase de entrenamiento se extraen a partir de los datos los parámetros de los modelos y en la fase de reconocimiento se utilizan estos modelos para verificar la pertenencia o no del objeto a la categoría representada por el modelo. El método paramétrico por excelencia es el método bayesiano [DH73], en el que el modelo es una distribución estadística. Así, en la fase de entrenamiento se estiman los parámetros de la distribución que siguen los datos de cada categoría y en la fase de reconocimiento se utilizan estas distribuciones para saber con qué probabilidad pertenece el objeto a una cierta categoría, teniendo también en cuenta la probabilidad a priori.

Los métodos de clasificación no paramétricos se caracterizan por la ausencia de un modelo basado en parámetros, ya que en este caso son los propios objetos proporcionados en la fase de entrenamiento (o prototipos) los que actúan como modelos. El método no

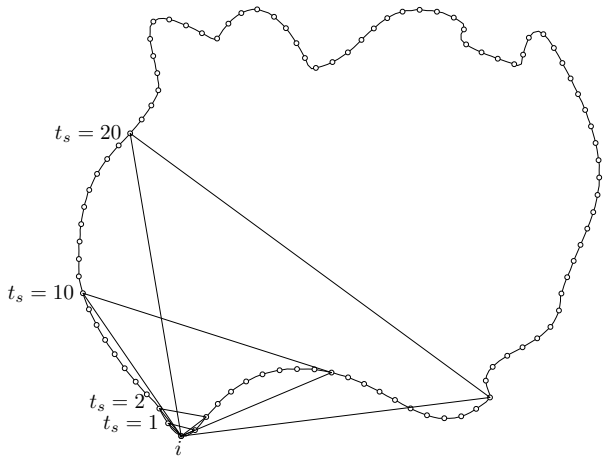


Figura 2.14: Ejemplo de triángulos para un punto i, a partir de una selección previa de puntos equidistante sobre la forma (véase la Sección 2.2.4). Se ilustran los triángulos para 4 valores de t_s, 1, 2, 10 y 20.

paramétrico más conocido y utilizado es la regla de los k vecinos más próximos (k *nearest neighbours*, k-NN). Es un método extremadamente simple pero que proporciona excelentes resultados y se basa en una medida de disimilitud o distancia entre los objetos. De esta manera, un objeto se clasifica en una categoría u otra según el valor de la distancia entre este y los prototipos. Cuando los objetos son representados con vectores numéricos, una medida de distancia muy usual es la distancia euclídea. No obstante, si los objetos se representan con cadenas, la elección de la medida de disimilitud es un tema más complicado, ya que estas no tienen una representación en el espacio euclídeo y, por tanto, requieren de la definición de otro tipo de distancias.

A continuación mencionaremos, sobre todo, distancias o medidas de disimilitud para ser utilizadas con los métodos no paramétricos. En cuanto a los métodos paramétricos sólo vamos a hacer mención de los modelos ocultos de Markov [Rab89], que son quizás el método probabilístico por antonomasia para el modelado estadístico de cadenas.

2.3.1. Distancias

Una distancia es una función que cuantifica la disimilitud entre dos patrones que describen una forma. De manera que, para reconocer formas bidimensionales, es necesario medir la disimilitud entre la forma y los prototipos de la base de datos.

Dado un conjunto de patrones, $S = \{x, y, z\}$, una distancia, $d : S \times S \to \mathbb{R}$, es una métrica si cumple las siguientes propiedades:

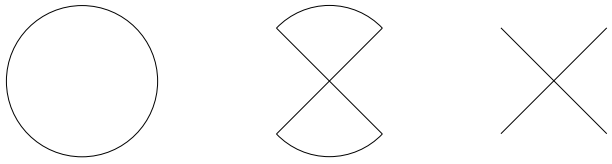
1. $d(x, y) = 0 \iff x = y$ (identidad),
2. $d(x, y) > 0$ si $x \neq y$,
3. $d(x, y) = d(y, x)$ (simetría),

4. $d(x, z) \leq d(x, y) + d(y, z)$ (desigualdad triangular).

El ser una métrica es útil en muchas aplicaciones. Sobre todo porque permite acelerar el reconocimiento con el indexado en bases de datos.

Una distancia que satisface las tres primeras propiedades y viola la desigualdad triangular se conoce como semimétrica.

Muchos estudios de psicología concluyen que la percepción humana no obedece las propiedades de una métrica [AG88, Tve77]. Un ejemplo es la simetría: la percepción humana no siempre encuentra que una forma a es similar a otra b del mismo modo en que b se parece a a. En particular, una variante a de un prototipo b se juzga frecuentemente más similar a b que viceversa [Tve77]. Por ejemplo, una elipse puede ser percibida de manera más similar a un círculo (puede ser un círculo estirado) que un círculo a una elipse [BCGJ95]. Además, la desigualdad triangular no encaja con cómo un humano compararía formas. Este hecho se ilustra en la Figura 2.15, donde las formas a y b son similares, es decir, en cierto modo $d(a, b)$ es pequeña. Del mismo modo, $d(b, c)$ es pequeña. Mientras que las formas a y c son muy diferentes, es decir, $d(a, c)$ es una distancia grande. Con lo que, $d(a, b) + d(b, c) < d(a, c)$, que viola la desigualdad triangular. Es por esto que algunos investigadores defienden que el uso de una métrica no es adecuado para el reconocimiento de formas [VH01].



(a) (b) (c)

Figura 2.15: Ejemplo que muestra cómo el juicio humano no atiende a la desigualdad triangular.

A continuación se comentan una serie de distancias para comparar conjuntos o secuencias de puntos, la mayoría de las cuales no son métricas. En lo que sigue, x e y serán dos conjuntos o cadenas de puntos en $\mathbb{R}^k$, siendo sus tamaños m y n, respectivamente.

2.3.1.1. Distancia de Minkowsky

Dados dos puntos x_i e y_j (de los conjuntos o cadenas x e y) pertenecientes a $\mathbb{R}^K$, la distancia de Minkowsky (o distancia L_p) se define como

$$d(x_i, y_j) = \left(\sum_{k=0}^{K} |x_{ik} - y_{jk}|^P \right)^{1/P}.$$

Para $P = 1$, se llama distancia de *Manhattan*. Para $P = 2$, se conoce como distancia Euclídea.

Si $m = n$ y la correspondencia entre los dos conjuntos de puntos se conoce, la distancia de Minkowsky se calcula entre los puntos y la distancia total entre las formas x e y es simplemente la suma de estas distancias. Sin embargo, en la mayoría de aplicaciones el número de puntos no suele ser el mismo y la correspondencia de estos no se conoce a priori. De este modo, el método de cálculo de la disimilitud debe de encontrar primero una correspondencia óptima.

2.3.1.2. Distancia de Hausdorff

La distancia de Hausdorff es la máxima distancia de un conjunto al punto más cercano del otro conjunto. Se suele utilizar cuando los dos conjuntos de puntos tienen tamaño diferente, con lo que no existe una correspondencia uno a uno. La distancia de Hausdorff asimétrica de x a y, $\overrightarrow{d}_H$, se define como

$$\overrightarrow{d}_H(x, y) = \max_{x_i} \min_{y_j} d(x_i, y_j).$$

Se puede ver que $\overrightarrow{d}_H(x, y)$ no es igual a $\overrightarrow{d}_H(y, x)$. Una definición más general es la siguiente:

$$d_H(x, y) = \max(\overrightarrow{d}_H(x, y), \overrightarrow{d}_H(y, x)).$$

Para un número finito de puntos, esta distancia se puede calcular utilizando diagramas de Voronoi con una complejidad temporal $O((m + n) \log(m + n))$ [ABB95].

Debido a que un sólo punto puede determinar el valor de la distancia, la distancia de Hausdorff es muy sensible al ruido. Para reducir este problema existen variantes de este método como la distancia de Hausdorff parcial o la distancia de Hausdorff de orden p [VH01].

2.3.1.3. Distancia de ángulo de la tangente

La distancia de ángulo de la tangente fue definida en [AcH$^+$91] para comparar funciones ángulo de la tangente (Sección 2.2.2.3) sobre aproximaciones poligonales de contornos:

$$d_T(x, y) = \left(\sum_i |\phi_x(i) - \phi_y(i)|^P \right)^{1/P},$$

donde los polígonos son escalados de manera que las cadenas de puntos tengan el mismo tamaño. Se seleccionará la P dependiendo de la distancia de Minkowsky que queramos utilizar.

En [LL00], para conseguir que esta distancia fuera más robusta a las distorsiones locales, la búsqueda de correspondencias en las funciones ángulo de la tangente se realiza permitiendo un estiramiento no uniforme de las partes.

2.3.1.4. Distancia de Fréchet

La distancia de Fréchet es una distancia entre curvas. Una explicación sencilla de la distancia de Fréchet es la siguiente. Supongamos que un hombre está paseando a su perro, el movimiento del hombre se describe con una curva y el del perro con otra. Ambos pueden controlar su velocidad, pero no se les permite volver hacia atrás. La distancia de Fréchet de las curvas es entonces, la mínima longitud de la correa necesaria.

Formalmente, sean $x(\alpha(l))$ e $y(\beta(l))$ dos curvas paramétricas y sean sus parametrizaciones α y β funciones continuas del mismo parámetro $l \in [0,1]$, tal que $\alpha(0) = \beta(0) = 0$ y $\alpha(1) = \beta(1) = 1$. La distancia de Fréchet es el mínimo sobre todas las parametrizaciones $\alpha(l)$ y $\beta(l)$ (monótonas crecientes) de la máxima distancia $d(x(\alpha(l)), y(\beta(l)))$ [VH01]:

$$d_F(x,y) = (\inf_{\alpha,\beta} \max_{l\in[0,1]} d(x(\alpha(l)), y(\beta(l))).$$

Según [Lem09], esta distancia es equivalente al alineamiento temporal no lineal (comentado en la Sección 2.3.4 y con mucho más detalle en la Sección 4.1) utilizando como distancia local la de Minkowsky, con $P = \infty$.

2.3.2. El método húngaro

El método húngaro [Mun57], también conocido como el algoritmo *Kuhn-Munkres*, es un algoritmo de optimización combinatoria que resuelve problemas de asignación en tiempo $O(m^2n)$. El algoritmo modela el problema de asignación como una matriz de costes $C^{m\times n}$, donde cada elemento C_{ij} representa el coste de asignar el trabajador i-ésimo al trabajo j-ésimo [Pil06]. Para comparar formas, los trabajadores serán los puntos de x y los trabajos los puntos de y y los costes, distancias entre puntos.

Es un proceso iterativo que comienza con la eliminación de Gauss-Jordan para que aparezcan ceros (al menos uno por línea y por columna). Si todas las filas tienen al menos una intersección con coste cero que no ha sido ocupada por otra fila, nos encontramos en el óptimo y termina el algoritmo. Si no, comienza un proceso de selección de filas y columnas, en base a los ceros, para obtener un parámetro δ que se suma o se resta a los elementos de la matriz dependiendo también de estas selecciones, volviendo en este punto de nuevo al principio.

Podemos encontrar una implementación en [Knu93].

Este método se utiliza en [BMP02] para comparar los *shape contexts* (véase la Sección 2.2.11.1).

2.3.3. Reconocimiento directo y técnicas de correlación

En este tipo de técnicas se utilizan las propias imágenes (normalmente imágenes binarias o en escala de grises) para realizar la comparación frente a unos prototipos. El método de comparación puede ser tan simple como una comparación píxel por píxel [GW92] o tan compleja como la combinación de varias medidas [Tub89]. Aunque estos métodos son intuitivos y tienen una sólida base matemática, suelen ser bastante sensibles al ruido. Un método alternativo que soluciona parcialmente este problema es la deformación de las

imágenes, donde la figura se deforma para buscar la correspondencia con las imágenes conocidas o prototipos de la base de datos [JZ97]. La medida de disimilitud se obtiene en función de la cantidad de deformación necesaria.

2.3.4. Comparación de cadenas

Dadas dos cadenas (formadas por cualquier representación del contorno), la comparación entre ellas es un proceso por el cual se mide cuánto difieren. En algunas ocasiones existe una correspondencia natural entre los componentes de ambas cadenas y la comparación se puede realizar utilizando alguna distancia de Minkowsky (Sección 2.3.1.1). Sin embargo, en la mayoría de situaciones esto no ocurre, porque ambas cadenas pueden haber sufrido algún tipo de corrupción. Es por esto que debe encontrarse la correspondencia apropiada sobre todas las correspondencias posibles. Para ello existen métodos eficientes basados en programación dinámica [SK83].

Los métodos de comparación de cadenas más conocidos son la distancia de edición y el alineamiento temporal no lineal (*dynamic time warping*). La distancia de edición ha sido muy utilizada para la comparación de formas [Mae91, MP98, SKK03, NB06]. No obstante, se han obtenido resultados mucho más competitivos (sino los mejores) en tareas de recuperación de formas (con el *bull-eye test*, Sección 2.5) con el alineamiento temporal no lineal junto con las representaciones de puntos con información global [AYV03, AO04, ARKF07] (Sección 2.2.11).

En los Capítulos 3 y 4 se hablará con más detalle sobre estas distancias.

2.3.5. Técnicas estructurales

Las técnicas de esta sección nos pueden servir para reconocer las representaciones estructurales mencionadas en la Sección 2.2.10.

Por un lado, tenemos las técnicas gramaticales, donde el reconocimiento se realiza mirando si la forma pertenece al lenguaje que representa una gramática [AU73]. A estos métodos también se les puede añadir propiedades estocásticas [SB85].

Por otro lado, tenemos las técnicas para grafos y árboles [Fu82, GT78] (en el caso de las cadenas, véase la Sección 2.3.4). La comparación entre grafos es el proceso que permite encontrar una correspondencia entre nodos y vértices que satisfagan ciertos criterios asegurando que las subestructuras de un grafo sean mapeadas en subestructuras similares del otro [CFSV04]. Una manera muy común de realizar esta correspondencia es usar operaciones de edición sobre grafos, incluyendo la inserción de nodos, el borrado de nodos y la sustitución de nodos. Después de asignar a cada operación un coste, se busca una secuencia de operaciones de coste mínimo. El coste total de esta secuencia es el coste de la correspondencia entre los grafos [GXTL, Bil05]. Estas técnicas tienen un elevado coste computacional, lo que lleva a tener que utilizar heurísticos.

2.3.6. Modelos ocultos de Markov

Uno de los métodos más utilizados para el modelado y reconocimiento de cadenas son los modelos ocultos de Markov [Rab89] y, hoy por hoy, están empezando a ser una

alternativa en el reconocimiento de formas bidimensionales [HK91, FMJ97, AYV00, CL01, BM04, BMF04, TGJ07].

Un modelo oculto de Markov se define como un proceso estocástico con dos mecanismos interrelacionados. Por un lado, una cadena de Markov que tiene un número finito de estados, y por el otro, un conjunto de funciones estocásticas, cada una asociada a un estado [Rab89]. En un instante discreto de tiempo, se asume que el proceso está en alguno de estos estados y una observación es generada por la correspondiente función estocástica. La cadena de Markov entonces cambia de estado de acuerdo a sus probabilidades de transición. Las observaciones pueden ser discretas o continuas.

Para utilizar estos modelos en tareas de clasificación, se construye un modelo para cada categoría de las formas, es decir, se entrena un modelo para que represente cada una de las categorías con muestras conocidas. Una vez hecho esto, para clasificar una muestra dada, se obtiene la probabilidad de que esta haya sido generada por cada modelo y así, la muestra queda clasificada en la categoría para la cual se haya obtenido la mayor probabilidad.

Los modelos ocultos de Markov se describirán con más detalle en el Capítulo 4.

2.4. La invarianza al punto inicial

Aunque no toda la literatura sobre reconocimiento de formas bidimensionales utiliza los contornos, la gran mayoría lo hace. Esta tesis está situada en este enfoque y en la comparación de estos contornos mediante métodos basados en cadenas, que como se ha mencionado en la Sección 2.3.4 han obtenido resultados muy competitivos en recuperación de formas. Sin embargo, si transformamos los contornos en cadenas y esta transformación es invariante a la rotación de la figura (por ejemplo, utilizando la curvatura, Sección 2.2.2.4) y los comparamos con estos métodos existe un problema: debemos encontrar un inicio adecuado en la codificación como cadena para realizar la comparación o para realizar el entrenamiento (en el caso de los modelos ocultos de Markov). Existen tres soluciones principales para este problema.

2.4.1. Elección de rotación de referencia y punto inicial

La idea de este enfoque es encontrar una rotación «canónica», y en base a esta rotación escoger el punto de inicio para calcular la distancia (o realizar el entrenamiento).

En dominios restringidos, se intenta encontrar una característica única para utilizar como punto de inicio. Por ejemplo, en el reconocimiento de perfiles humanos se suele utilizar la nariz u otra parte significativa del perfil. No obstante, se requieren métodos específicos para encontrar estas características. Incluso en un dominio tan restringido y bien entendido como los perfiles humanos, buscar la nariz no es un problema trivial.

En dominios no restringidos, se suele utilizar un alineamiento que depende de algún valor de orientación, como por ejemplo, el eje mayor (Sección 2.2.1). Este método puede ser útil para dominios limitados en los que el eje mayor está bien definido, pero no es adecuado cuando puede existir ruido ya que es muy sensible a este [Hor86, JKS95]. Con los modelos ocultos de Markov, también se aplicó este criterio en [HK91]. Por otro lado, también existe la posibilidad de utilizar los descriptores de Fourier (Sección 2.2.7), como

en [BCP05], para encontrar el punto de inicio. Sin embargo, aparte de tener problemas de ambigüedad al realizar la orientación [FS02], este método también puede presentar problemas con formas que tienen una elipse principal (la que modela la primera frecuencia) cercana a la circularidad, como veremos en la Sección 6.1.

2.4.2. Características invariantes a la rotación

Muchos trabajos de la literatura consiguen la invarianza al punto inicial simplemente utilizando características invariantes a la rotación [CGK03], como la proporción entre el perímetro y el área, elongación, circularidad, mínima/máxima curvatura, curvatura media, entropía, perímetro de la caja circundante, etc. Aparte de todas estas características también están los momentos [Hu62] y el espectro de Fourier (Sección 2.2.7) utilizado en [ZL05] y en [CL01] (haciendo uso de los modelos ocultos de Markov). Todos estos enfoques ofrecen una solución rápida y sencilla. Sin embargo, todos ofrecen una precisión un tanto pobre. Para obtener esta invarianza, todos los datos que contiene este tipo de información deben ser descartados, y por tanto, inevitablemente mucha información útil para discriminar entre formas se descarta en el proceso.

Todas estas características pueden ser útiles para discriminar entre formas que apenas se parezcan, es decir, una discriminación gruesa. Como se comenta en la Sección 2.2.1 estas características nos pueden servir como un filtro muy rápido para descartar formas. Pero si queremos una discriminación más fina tendremos que utilizar otros métodos.

2.4.3. Alineamiento por fuerza bruta de cadenas cíclicas

Existen multitud de trabajos que admiten que los métodos anteriores para conseguir la invarianza al punto inicial no son satisfactorios para la mayoría de los dominios, y consiguen esta invarianza con una búsqueda por fuerza bruta para todos los posibles puntos iniciales de las cadenas a comparar, a expensas de incrementar el coste computacional [AO03, AO04, ARKF07, AYV03, GW99]. Por ejemplo, en [AO03] (comentado en la Sección 2.2.8.1) se comenta que mientras que con su método otro tipo de invarianzas se consiguen de manera trivial con su representación, el conseguir la invarianza al punto inicial sólo se puede obtener teniendo en cuenta todos los posibles desplazamientos circulares de la cadena. Esto les lleva a tener que utilizar un algoritmo con coste $O(n^3)$. Por este motivo, estos trabajos además ofrecen la posibilidad de mitigar este elevado coste computacional, con soluciones aproximadas, como por ejemplo, buscando un número pequeño de posibles puntos iniciales [AO04, GW99]. Obviamente, estas soluciones son heurísticos similares a los comentados en la Sección 2.4.1, y por tanto, se deben ajustar dependiendo de la aplicación.

Llegamos a la conclusión pues, de que si queremos conseguir una invarianza total al punto inicial tendremos que hacer uso de la fuerza bruta. Es aquí donde surge el concepto de cadena cíclica. Los contornos no se pueden representar con simples cadenas, sino con cadenas cíclicas, es decir, cadenas que no tienen principio ni final claramente definido. De este modo, comparar dos cadenas cíclicas es equivalente a comparar las cadenas utilizando todo posible punto inicial de estas.

Esta comparación entre todo posible punto inicial, como hemos dicho, resulta muy costosa computacionalmente. Existen trabajos orientados a paliar este coste, sobre todo en el terreno de las distancia de edición (véase la Sección 2.3.4), como veremos en el Capítulo 3. En el Capítulo 4, veremos qué soluciones existen a este respecto, en la literatura, relacionadas con el alineamiento temporal no lineal (Sección 2.3.4) y los modelos ocultos de Markov (Sección 2.3.6) .

2.5. Métodos de evaluación

La evaluación de un sistema de reconocimiento es crucial. Por supuesto, esta evaluación depende de la aplicación que queramos darle al sistema [Car01]. No obstante, los investigadores han creado y utilizado muchos métodos para este fin. Seguramente el método más utilizado para evaluar sistemas de clasificación es la tasa de acierto (o de error), es decir, el porcentaje de aciertos (o de errores) que se producen con ese sistema para unos corpus dados.

En el marco de la recuperación de formas[4], es el gráfico *precision/recall* (PR) [Rij79, MMS+01, MRS08]. Los gráficos PR nos ofrecen una manera de testear y comparar el rendimiento de diferentes sistemas en sólo un gráfico. Para calcular este gráfico se utilizan las siguientes fórmulas:

$$Precision = \frac{número\ de\ imágenes\ recuperadas\ relevantes}{número\ de\ imágenes\ recuperadas},$$

$$Recall = \frac{número\ de\ imágenes\ recuperadas\ relevantes}{número\ de\ imágenes\ relevantes}.$$

El gráfico PR muestra entonces la *precision* en función del *recall* para una imagen de test. En el caso de que queramos obtener un gráfico PR para varias muestras, se fijan niveles de *recall* y se hace la media para todos las muestras en esos niveles. Para más detalle puede consultarse [Rij79, MRS08].

Aunque estos gráficos son muy útiles, en ocasiones es necesario poder comparar los sistemas recuperadores de formas con un solo valor. El MPEG-7 [Bob01] ha popularizado una serie de tests con bases de datos estándar para la evaluación de los sistemas. Aunque el más conocido es el *bull-eye test* [Bob01] utilizado por la inmensa mayoría de la literatura dedicada a la recuperación para comparar métodos. Este test consiste en un porcentaje para un determinado *recall*, cuanto más alto es este el método supuestamente será mejor. Esta medida es algo así como un solo punto en la curva PR (no es así, es para hacernos una idea), lo cual en algunos casos podría no tener una correlación con dicha curva.

En la comunidad de recuperación de información ha proliferado la *mean average precision* (MAP) [MRS08] la cual ha demostrado ser una medida de calidad, con una buena discriminación y mucha estabilidad. MAP se calcula de la siguiente forma. Si las imágenes

[4]En general, un sistema que funcione bien en recuperación de formas lo hará también en otro tipo de tareas de reconocimiento como la clasificación.

relevantes para una muestra de test $q_j \in Q$ son $\{d_1, \ldots, d_{m_j}\}$ y R_{jk} son las imágenes recuperadas ordenadas desde la primera (imagen recuperada) hasta que recuperas la imagen d_k, entonces

$$MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} Precision(R_{jk}).$$

Para una imagen de test, MAP aproxima el área bajo la curva PR. Así, para un conjunto de imágenes de test, MAP se acerca (de una manera abrupta) a la media de las áreas.

Por último, para evaluar la eficiencia en cuanto a tiempo, evidentemente, podemos utilizar la complejidad temporal [CLR90] y las propias medidas de tiempo de ejecución u otras medidas relativas a partes críticas de los algoritmos que nos pueden hacer ver el ahorro temporal.

En algunos casos la precisión puede estar reñida con el tiempo necesario para realizar la representación y el reconocimiento. Se puede tener un método muy preciso pero que no se puede utilizar en tiempo real. Todo depende de la aplicación que queramos darle. En ocasiones necesitaremos tiempo real, con lo que podríamos tener que renunciar a precisión, y en otras ocasiones nos sirve un método *off-line*. Por otro lado, es posible también, que para una aplicación muy concreta, un método simple (con unas tasas de clasificación no muy altas en bases de datos complicadas, como las del MPEG-7) sea suficiente.

2.6. Discusión

En este capítulo se ha pretendido dar una visión de las etapas de parametrización y reconocimiento de los sistemas reconocedores de formas bidimensionales con los contornos y los métodos de evaluación que se utilizan en la literatura. Se ha comentado también el problema de la invarianza al punto de inicio, siendo este un problema importante con este tipo de descriptores, argumentando que la utilización de cadenas cíclicas es la única forma de conseguir esta invarianza.

Hemos destacado en estos descriptores, las cadenas con información global de cada punto, y en los reconocedores, las técnicas de programación dinámica para comparar estas cadenas y los modelos ocultos de Markov, que son precisamente los descriptores y técnicas que vamos a considerar en esta tesis para tratar el problema de la invarianza al punto de inicio.

En el siguiente capítulo se hablará de la distancia de edición cíclica para que el lector se de cuenta de la naturaleza del problema de las cadenas cíclicas y de cómo es tratado en este caso particular. La mayoría de las aportaciones de esta tesis parten de ahí.

Capítulo 3

Distancia de edición

Porco Rosso: «*A pig that doesn't fly is just a pig.*»
Hayao Miyazaki, *Pòrco Rósso* (1998).

Cuantificar el grado de semejanza entre dos cadenas es un problema muy importante en el reconocimiento de formas. La distancia de edición es quizás una de las distancias más conocidas para este fin. Como se comentó en la Sección 2.4, las cadenas estándar no son adecuadas cuando tratamos de modelar cadenas de contornos, ya que en esos casos no existe un inicio o fin claro de la cadena. Es por esto que resulta más conveniente el uso del concepto de cadena cíclica. En este capítulo repasaremos las técnicas para medir la distancia de edición entre cadenas y las extensiones a cadenas cíclicas.

3.1. Distancia de edición

Estamos interesados en saber si dos cadenas son muy semejantes o no, y en cuantificar dicho grado de semejanza. Si comparamos dos cadenas, se define la distancia de edición (DE) [WF74] entre estas como el coste mínimo de la transformación de una cadena a la otra mediante operaciones de edición. Estas operaciones de edición normalmente son la inserción, el borrado y la sustitución de un símbolo por otro.

La DE puede ser calculada con los algoritmos presentados en [Lev66, WF74, Sel80, MP80] y se ha utilizado a problemas de corrección de errores, reconocimiento de patrones y otras aplicaciones relacionadas [HD80, Fu82, SK83, Nav01] y, por supuesto, al reconocimiento de formas bidimensionales [Mae91, MP98, SKK03, NB06], como se menciona en la Sección 2.3.4.

De manera formal. Sea Σ un alfabeto de talla finita, y Σ^* el conjunto de cadenas de longitud finita que pueden construirse con Σ. El símbolo λ denota la cadena o vacía. Sean $x = x_1 x_2 \dots x_m$, $x_i \in \Sigma$, $1 \leq i \leq m$ e $y = y_1 y_2 \dots y_n$, $y_j \in \Sigma$, $1 \leq j \leq n$, no siendo ninguna nula. Indicaremos con $x_{i:j}$ la subcadena $x_i x_{i+1} \dots x_j$. Adoptaremos el convenio de que $x_{i:j} = \lambda$ cuando $i > j$.

Las operaciones de edición elementales que permiten transformar x en y son de tres tipos:

- Inserción: el símbolo y_j se inserta en la cadena x ($\lambda \mapsto y_j$),
- Borrado: el símbolo x_i se elimina de la cadena x ($x_i \mapsto \lambda$),
- Sustitución: el símbolo x_i se sustituye por el símbolo y_j ($x_i \mapsto y_j$).

Una transformación de edición de x en y consiste en una lista de operaciones de edición elementales. Por ejemplo, en el siguiente ejemplo se muestra una posible transformación de '10101' en '1100':

$$10101 \xrightarrow{1\mapsto 1} 10101 \xrightarrow{0\mapsto \lambda} 1101 \xrightarrow{1\mapsto 1} 1101 \xrightarrow{0\mapsto 0} 1101 \xrightarrow{1\mapsto 0} 1101$$

Una transformación de edición entre dos cadenas puede visualizarse mejor como un camino en un grafo de edición[1]. Un grafo de edición de x e y, $G_E(x, y)$, es un grafo dirigido acíclico basado en una malla de $(m+1)\cdot(n+1)$ vértices (i, j), $0 \leq i \leq m$, $0 \leq j \leq n$. Los arcos de $G_E(x, y)$ se dividen en tres tipos, correspondientes a los tres tipos de operaciones de edición:

- Arcos horizontales: $\{((i-1, j), (i, j)) \mid 0 < i \leq m, 0 \leq j \leq n\}$ (borrados),
- Arcos verticales: $\{((i, j-1), (i, j)) \mid 0 \leq i \leq m, 0 < j \leq n\}$ (inserciones),
- Arcos diagonales: $\{((i-1, j-1), (i, j)) \mid 0 < i \leq m, 0 < j \leq n\}$ (sustituciones o aciertos).

En la Figura 3.1 se muestra un grafo de edición para las cadenas del ejemplo anterior.

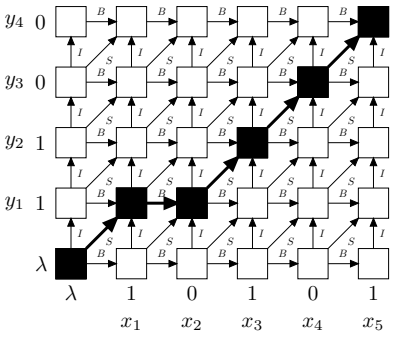


Figura 3.1: Grafo asociado a la comparación de las cadenas 10101 y 1100, con un posible camino de edición.

Una transformación entre x e y se representaría en un grafo como un camino de edición en $G_E(x, y)$, que partiría de $(0, 0)$ y llegaría a (m, n). En la Figura 3.1, se muestra el camino asociado a la transformación del ejemplo anterior, indicando además los arcos correspondientes a borrados (D), inserciones (I) y sustituciones (S).

[1]En el caso de que la transformación tenga una traza asociada correspondiente [MV93]. Son estas transformaciones las que nos pueden llevar a caminos de edición óptimos, que son los que nos interesan como veremos a continuación.

Para asignar valores numéricos a las transformaciones de edición que reflejen el coste que llevan asociado, es necesario definir costes para cada una de las operaciones de edición. Aunque estos costes pueden variar en función de los símbolos implicados, solo consideraremos costes de inserción (γ_I), de borrado (γ_D), de sustitución (γ_S) y de acierto (γ_M). Una posible elección para los costes sería $\gamma_I = \gamma_D = \gamma_S = 1$ y $\gamma_M = 0$. Con ello, la suma de los costes de una transformación de edición (o camino en un grafo) proporcionaría el número de operaciones de error. Esta elección de pesos da lugar a la denominada distancia de Levenshtein [Lev66]. Es posible definir funciones de ponderación de las operaciones de edición que dependen de los símbolos implicados, pero en aras de la claridad, desarrollaremos la explicación con pesos constantes para cada tipo de operación de edición.

Se define la distancia de edición[2] entre dos cadenas como el peso del camino de edición de menor coste en $G_E(x, y)$. Dicho de otra forma, si llamamos $\mathcal{P}$ al conjunto de caminos en el grafo $G_E(x, y)$, y $W_\gamma(p)$ la suma de los pesos de los arcos al camino $p \in \mathcal{P}$, utilizando el vector de pesos $\gamma = (\gamma_I, \gamma_D, \gamma_S, \gamma_M)$, entonces,

$$DE(x, y, \gamma) = \min_{p \in \mathcal{P}} W_\gamma(p).$$

Para el cálculo de $DE(x, y, \gamma)$ se utiliza un procedimiento de programación dinámica basado en la siguiente relación recursiva:

$$DE(x_{1:i}, y_{1:j}, \gamma) = \min \begin{cases} DE(x_{1:i-1}, y_{1:j}, \gamma) + \gamma_D, \\ DE(x_{1:i}, y_{1:j-1}, \gamma) + \gamma_I, \\ DE(x_{1:i}, y_{1:j-1}, \gamma) + \gamma_S \cdot \delta_K(x_i y_j) + \gamma_M(1 - \delta_K(x_i y_j)) \end{cases}$$

siendo δ_K la función delta de Kronecker. La implementación de este procedimiento lleva a un algoritmo para el cálculo de $DE(x, y, \gamma)$ con una complejidad temporal de $O(mn)$ y espacial de $O(\min(m, n))$ [WF74].

3.2. Distancia de edición normalizada

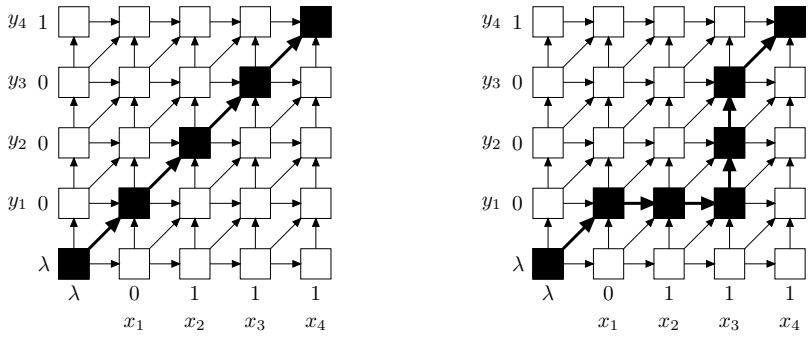
La distancia de edición no resulta siempre apropiada, al carecer de una cierta normalización al respecto de la talla de los objetos comparados. Así, parece sensato considerar que un error en la comparación de dos cadenas de longitud 3, es mucho más importante que un error en la comparación de dos cadenas de longitud 1000.

Si llamamos $L(p)$ al número de operaciones de edición (o arcos en el grafo) asociadas al camino p, entonces se define la distancia de edición normalizada $DEN(x, y, \gamma)$ como

$$DEN(x, y, \gamma) = \min_{p \in \mathcal{P}} \frac{W_\gamma(p)}{L(p)}.$$

[2] Si la función γ cumple la desigualdad triangular este camino (traza) también minimiza la secuencia de transformaciones [WF74].

$DEN(x, y, \gamma)$ no está definida cuando ambas cadenas están vacías, en cuyo caso $L(p) = 0$. Además, no es posible obtener $DEN(x, y, \gamma)$ calculando previamente $DE(x, y, \gamma)$ y después normalizando, tal y como se muestra en 3.2. Tampoco puede obtenerse mediante el cálculo de distancias de edición normalizadas localmente [MV93]. Así pues, la utilización del algoritmo de Wagner y Fischer no es factible para el cálculo de la DEN.



(a) (b)

Figura 3.2: (a) Camino óptimo entre las cadenas $x = 0111$ y $y = 0001$, con $\gamma = (2, 2, 3, 0)$. (b) Camino óptimo para la distancia de edición normalizada. Nótese que $DEN(x, y, \gamma) = 1{,}33$ es menor que la distancia de edición aplicando después una normalización (según la longitud del camino), $\frac{DE(x,y,\gamma)}{4} = 1{,}5$.

La idea básica para el de la DEN consiste en comprobar que sólo un conjunto de longitudes de caminos de edición son posibles, de forma que podemos calcular un camino óptimo para cada una de las longitudes. Una vez calculado un camino óptimo para cada longitud posible, dividimos su coste por la longitud y calculamos el mínimo valor del coste. En [MV93], se muestra cómo hacerlo. Utilizando las relaciones recursivas de los Teoremas 4.1 [MV93] y 4.2 [MV93] podemos calcular $DE(x, y, k)$ mediante programación dinámica para todos los valores de k válidos según el Lema 4.1 [MV93]. Esto permite implementar el cálculo de $DEN(x, y)$ con una complejidad temporal $O(mn \cdot \text{mín}(m, n))$ [MV93].

La DEN presenta la característica de no ser formalmente una métrica, ya que la desigualdad triangular no siempre se cumple. Se pueden encontrar ejemplos en los que se muestra este hecho para ciertos pesos de γ en los cuales una inserción o borrado tiene un peso mucho menor que cualquier otro peso del vector [MV93].

En [VMA95] se describe un algoritmo menos costoso utilizando técnicas de programación fraccional, más concretamente los autores se basaron en el algoritmo de Dilkenbach para obtener un método más rápido para calcular la DEN.

El problema a ser solucionado es el siguiente,

Problema N.

$$d^* = \min_{p \in \mathcal{P}} \frac{W(p)}{L(p)}, \quad W, L : \mathcal{P}; \quad L(p) > 0, \ \forall p \in \mathcal{P}.$$

El Problema N se puede resolver por medio del problema $N(\lambda)$:

Problema $N(\lambda)$**.**

$$d^*(\lambda) = \min_{p\in\mathcal{P}}(W(p) - \lambda L(p)), \quad W, L : \mathcal{P}; \quad L(p) > 0, \ \forall p \in \mathcal{P}.$$

El algoritmo puede empezar con una inicialización cualquiera para λ. En particular resulta cómoda la elección del camino de edición de la distancia no normalizada. Sin embargo, es conveniente que el proceso iterativo comience en un punto cercano a la solución final, para acelerar la convergencia. El problema $N(\lambda)$ se resuelve entonces obteniendo un camino óptimo p^*. Con este valor de p^*, un nuevo valor de λ es calculado con $\frac{W(p^*)}{L(p^*)}$. El procedimiento se repite hasta que se llega a la convergencia. El algoritmo de la Figura 3.3 muestra el procedimiento. Se puede demostrar que el cálculo de $\min_{p\in\mathcal{P}}(W(p) - \lambda' L(p))$ puede ser formulado en términos de la distancia de edición utilizando otros costes:

$$\min_{p\in\mathcal{P}}(W(p) - \lambda' L(p)) = DE(x, y, \gamma') - \lambda'(m + n), \quad \text{siendo } \gamma' = (\gamma_I, \gamma_D, \gamma_S + \lambda', \gamma_M + \lambda').$$

Figura 3.3: Algoritmo DEN.

Entrada: $x, y : \Sigma^*$
Salida: $\lambda^* : \mathbb{R}$
Inicio
  $p^* =$ `ElementoArbitrario(`$\mathcal{P}$`)`
  $\lambda^* = \frac{W(p^*)}{L(p^*)}$
  repite
    $\lambda' = \lambda^*$
    $p^* = \arg\min_{p\in\mathcal{P}}(W(p) - \lambda' L(p))$
    $\lambda^* = \frac{W(p^*)}{L(p^*)}$
  hasta $\lambda^* = \lambda'$
  devolver λ^*
Fin

El Teorema 2 [VMA95] garantiza que tras un mínimo de iteraciones se converge al valor de la distancia de edición normalizada. De este modo, el algoritmo tiene un coste computacional de $O(tmn)$, siendo t el número de iteraciones. En [VMA95], se muestra con experimentos que el valor suele estar entre 2 y 4.

3.3. Distancia de edición cíclica

Sea $\rho : \Sigma^* \times \mathbb{N} \to \Sigma^*$ una función de desplazamiento cíclico:

$$\rho(x) = x_2 \dots x_m x_1.$$

Sea $\rho^k(x) = \overbrace{\rho(\rho(\ldots\rho}^{k \text{ veces}}(x)\ldots)) = x_{k+1:m}x_{1:k}$ y $\rho^0(x) = x$.

Dos cadenas x y x' son cíclicamente equivalentes si $x = \rho^k(x')$, para algún k. La función ρ define una relación de equivalencia en Σ^*:

$$x \equiv x' \Longleftrightarrow \exists i, j : \rho^i(x) = \rho^j(x').$$

La clase de equivalencia de x es $[x] = \{\rho^k(x) : 0 \leq k < m\}$, formada por las cadenas equivalentes a x. La clase de equivalencia de una cadena recibe el nombre de cadena cíclica. Cualquiera de los miembros de esta clase de equivalencia es una cadena representativa de la cadena cíclica. Por ejemplo, sean $0 \in \Sigma$ y $1 \in \Sigma$, el conjunto $\{0111, 1110, 1101, 1011\}$ es una cadena cíclica y 1101 (o cualquiera de los otras cadenas en el conjunto) puede hacer de representante.

Definimos la distancia de edición entre dos cadenas cíclicas como

$$DEC([x],[y]) = \min_{x' \in [x]} \min_{y' \in [y]} DE(x', y') = \min_{0 \leq i < m} \min_{0 \leq j < n} DE(\rho^i(x), \rho^j(y)).$$

Definimos la distancia de edición entre una cadena cíclica y una cadena como

$$DEC([x], y) = \min_{0 \leq k < m} DE(\rho^k(x), y).$$

3.3.1. El algoritmo de fuerza bruta

En [Mae90] se demostró:

Lema 3.1 *Para cualquier representante y de $[y]$ tenemos*

$$DEC([x],[y]]) = DEC([x], y).$$

□

Como calcular $DEC([x],[y])$ se reduce a calcular $DEC([x], y)$, podemos calcular directamente $\min_{0 \leq k < m} DE(\rho^k(x), y)$ evaluando $DE(\rho^k(x), y)$ para todo k entre 0 y $m-1$ y seleccionando el menor valor (véase la Figura 3.4).

Este método de cálculo presenta una complejidad espacial $O(n)$ y temporal $O(m^2 n)$.

3.3.2. Algoritmo divide y vencerás

En [Mae90], se propuso una solución basada en la estrategia algorítmica de divide y vencerás [CLR90].

El algoritmo se basa en este resultado:

Teorema 3.2

$$DEC([x],[y]) = DEC([x], y) = DE_m(x \cdot x, y)$$

donde $DE_m(x \cdot x, y)$ es la distancia de edición entre cualquier subcadena de $x \cdot x$ de talla m y la cadena y (siendo $x \cdot x$ la concatenación de x consigo misma). □

Figura 3.4: Algoritmo de fuerza bruta.

Entrada: $x, y : \Sigma^*$
Salida: $d^* : \mathbb{N}$
Inicio
  $d^* = DE(x, y)$
  para $k = 1$ **hasta** $m - 1$ **hacer**
    $d^* =$ mín$(d^*,\ DE(\rho^k(x),\ y))$
  devolver d^*
Fin

Dadas x e y, cadenas representantes de las cadenas cíclicas $[x]$ y $[y]$, respectivamente, sea $G_{EE}(x, y)$ el grafo de edición extendido de x e y (véase la figura 3.5).

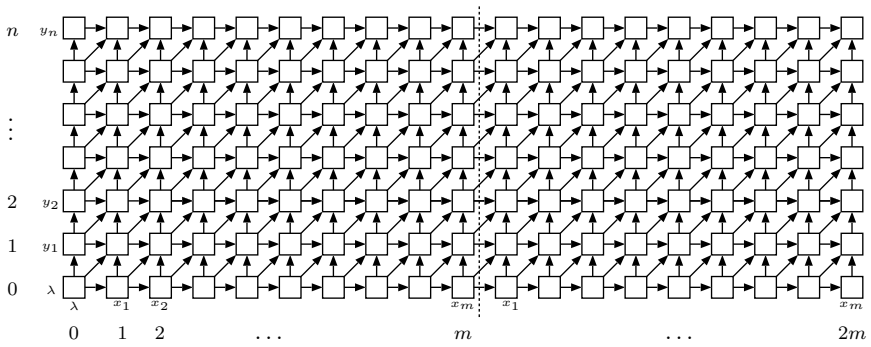


Figura 3.5: Grafo de edición extendido.

Al calcular $DE(\rho^k(x), y)$ buscamos el camino de peso mínimo en $G_{EE}(x, y)$ que parte del nodo $(k, 0)$ y finaliza en el nodo $(k{+}m, n)$. Los arcos horizontales y verticales pesan γ_D y γ_I, respectivamente, y los arcos diagonales pesan γ_S cuando $x_i = y_i$ y γ_M en caso contrario, siendo (i, j) las coordenadas del nodo al que apunta el arco (distancia de Levenshtein). Teniendo en cuenta que estos caminos de peso mínimo (caminos óptimos) no pueden cruzarse [Mae90], podemos seguir la aproximación que se representa en la Figura 3.6 para una cadena x de tamaño 8.

Primero se obtienen $P(0)$ y $P(m)$ (nótese que $P(m)$ es $P(0)$ desplazado m unidades a la derecha) al calcular la distancia de edición entre x e y. A continuación se calcula $P(c)$, siendo c el punto medio entre 0 y m, aprovechando que el camino debe discurrir entre $P(0)$ y $P(m)$ y, por tanto, no debe visitarse ningún nodo en la zona gris oscuro. Tampoco los nodos en la zona gris claro deben visitarse por razones obvias. A esta zona entre caminos la llamaremos canal. Una vez se conoce $P(c)$, se procede recursivamente buscando, por una parte, $P(c')$, para c' igual al punto medio entre 0 y c, aprovechando que debe discurrir entre el canal de $P(0)$ y $P(c)$ para realizar el menor número de cálculos; y por otra parte, $P(c'')$, para c'' igual al punto medio entre c y m, aprovechando que debe discurrir entre el canal de $P(c)$ y $P(m)$ para realizar el menor número de cálculos.

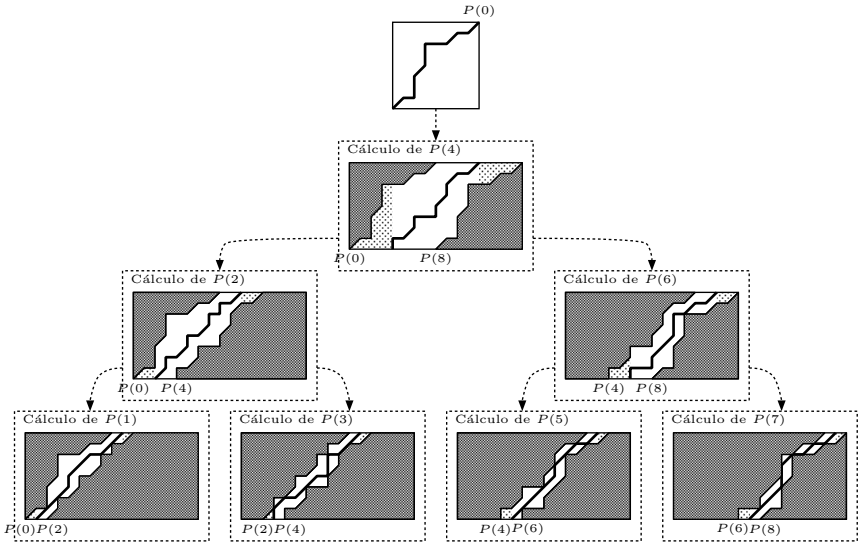


Figura 3.6: Árbol generado por el algoritmo de Maes.

En [Mae90] se presentó una versión iterativa de este procedimiento de cálculo. Presentamos aquí una versión recursiva que pone de manifiesto que el algoritmo sigue la estrategia algorítmica divide y vencerás [CLR90] (véase la Figura 3.7).

La complejidad temporal se puede deducir estudiando el gráfico que ilustra el método de cálculo (Figura 3.6). En cada uno de los grafos sobre los que se busca un camino óptimo sólo se visitan los nodos de la región blanca. La suma del número de nodos visitados en todos los grafos a una misma altura del árbol mostrado en la gráfica es $O(mn)$. El número de niveles del árbol es $O(\log m)$. El coste temporal del algoritmo es, pues, $O(mn \log m)$ y la complejidad espacial es $O(mn)$.

3.3.3. Algoritmo de ramificación y poda

El algoritmo de Maes puede mejorarse en la práctica si se adopta un enfoque de ramificación y poda, como se mostró en [MB00]. En ramificación y poda se propone la búsqueda del elemento de un conjunto Q que hace mínimo el valor de una función objetivo $f : Q \to \mathbb{R}$.

En nuestro problema el espacio de búsqueda es el conjunto $Q = \{0, 1, 2, \ldots, m-1\}$. La función objetivo es $f(k) = DE(\rho^k(x), y)$. Con $]l, r[$ se representará el estado/subconjunto $\{l+1, \ldots, r-1\}$. Un estado contiene un solo elemento si $l = r$. La función ramifica se define así:

$$\text{ramifica}(]l, r[) = \{]l, k[,]k-1, k+1[,]k, r[\},$$

donde $k = l + \lceil \frac{r-l}{2} \rceil$.

Sin tener en cuenta la definición de una función de cota inferior g para descartar canales, podemos definir el algorítmico para obtener una solución al problema de la distancia de

Figura 3.7: Algoritmo divide y vencerás.

Entrada: $x, y : \Sigma^*$
Salida: $d^* : \mathbb{N}$
var P: vector $[0..m]$ de caminos de edición
Inicio
  Sea d^* la distancia de edición entre x e y
  Sea $P[0]$ el camino de edición óptimo entre x e y
  Sea $P[m]$ igual $P[0]$ desplazado m pasos a la derecha
  si $m > 1$ **entonces**
    $d^* =$ `mín(`d^*`, SiguientePaso(`$x \cdot x$`, `y`, 0, `m`))`
  devolver d^*
Fin

función `SiguientePaso(`$X : \Sigma^*$`, `$y : \Sigma^*$`, `$l : \mathbb{N}$`, `$r : \mathbb{N}$`):`$\mathbb{N}$
Inicio
  $c = l + \lceil \frac{r-l}{2} \rceil$
  Sea d la distancia de edición entre $X_{c:c+m}$ conocidos $P[l]$ y $P[r]$
  si $l + 1 < c$ **entonces**
    $d =$ `mín(`d`, SiguientePaso(`X`, `y`, `l`, `c`))`
  si $c + 1 < r$ **entonces**
    $d =$ `mín(`d`, SiguientePaso(`X`, `y`, `c`, `r`))`
  devolver d
Fin

edición para cadenas cíclicas (véase la Figura 3.8).

Figura 3.8: Algoritmo de ramificación y poda

Entrada: $x, y : \Sigma^*$
Salida: $d^* : \mathbb{N}$
aux $g : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$
var $\mathcal{S} : 2^{\mathbb{N}\times\mathbb{N}}$; P : vector $[0..m$ de caminos de edición
Inicio
  Sea d^* la distancia de edición entre x e y
  Sea $P[0]$ el camino de edición óptimo entre x e y
  Sea $P[m]$ igual $P[0]$ desplazado m pasos a la derecha
  $\mathcal{S} = \{]0, m[\}$ *//Inicialización conjunto de estados*
  mientras $\mathcal{S} \neq \emptyset \wedge d^* > \min_{]l,r[\in\mathcal{S}} g(]l, r[)$ **hacer**
    $]l, r[= \arg\min_{]l',r'[\in\mathcal{S}} g(]l', r'[)$; $\mathcal{S} = \mathcal{S} - \{]l, r[\}$ *//Selección*
    //Ramificación
    $k = l + \lceil \frac{r-l}{2} \rceil$
    $d^* = \min(d^*, DE(\rho^k(x), y))$ (calcular $DE(\rho^k(x), y)$ entre $P[l]$ y $P[r]$)
    Sea $P[k]$ el camino óptimo de edición (subproducto de $DE(\rho^k(x), y)$)
    si $k > l + 1 \wedge g(]l, k[) < d^*$ **entonces**
      $\mathcal{S} = \mathcal{S} \cup \{]l, k[\}$
    si $r > k + 1 \wedge g(]k, r[) < d^*$ **entonces**
      $\mathcal{S} = \mathcal{S} \cup \{]k, r[\}$
  devolver d^*
Fin

Sea cual sea la cota inferior utilizada, el algoritmo de ramificación y poda efectúa, en el peor de los casos, el mismo número de cálculos de distancia de edición que el algoritmo de Maes, pues explora el mismo espacio, sólo que en un orden posiblemente diferente. Así pues, el coste achacable al cálculo de distancias de edición es $O(mn \log m)$. Veamos, pues, el coste que supone la gestión de $\mathcal{S}$. El número de estados que ingresan en $\mathcal{S}$ es, en el peor de los casos, $2m-1$. Simultáneamente no hay más de m estados en $\mathcal{S}$. Si $\mathcal{S}$ se implementa mediante un *heap*, el coste de cada inserción o extracción de un estado es $O(\log m)$. La gestión de $\mathcal{S}$ supone, pues, un coste temporal adicional $O(m \log m)$ y espacial $O(m)$. Sobre cada estado resultante de una ramificación debe evaluarse la función de cota inferior. La complejidad computacional del algoritmo será $O(mn \log m + m(2\omega + \log m)) = O(m(\omega + n \log m))$, donde ω representa el coste de evaluar la función de cota inferior.

Aunque en [MB00] se mencionan otras cotas, la que obtiene mejores resultados en la práctica es la siguiente:

$$g(]l, r[) = \text{máx}\left(0, \frac{DE(\rho^l(x), y) + DE(\rho^r(x), y) - (\gamma_D + \gamma_I)(r - l)}{2}\right) \leq \min_{l<k<r} DE(\rho^k(x), y).$$

Esta cota se calcula en tiempo $O(1)$. Por tanto, la complejidad computacional del

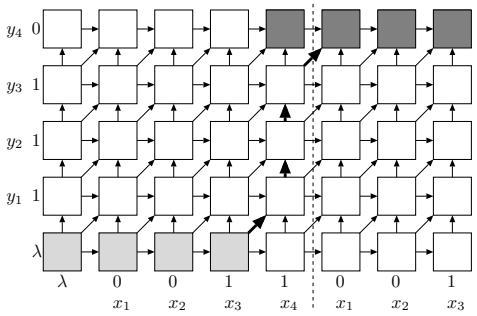


Figura 3.9: Cálculo del BB en el grafo de edición extendido para $x = 0011$ e $y = 1110$. Los aciertos tienen un peso 0 y cualquier otra operación tiene peso 1. El algoritmo BB obtiene 1 como un valor aproximado de $CED([x],[y])$ y su camino se muestra con flechas gruesas. El camino debe de empezar en cualquier nodo gris claro (fila inferior) y terminar en un nodo de color gris oscuro (fila superior).

algoritmo usando esta cota es $O(mn \log m)$.

3.3.4. El algoritmo de Bunke y Bühler

Si las soluciones anteriores para acelerar el cálculo no nos satisfacen todavía, podemos recurrir a soluciones aproximadas como el algoritmo de Bunke y Bühler [BB93].

Este algoritmo aproxima la CED buscando, en el grafo extendido, el camino óptimo que parte de cualquiera de los nodos $(i, 0)$ para $0 \leq i < m$, y llega a cualquiera de los nodos (i, n), para $m \leq i < 2m$ [JMPP04], sin ningún control sobre la longitud de la subcadena de $x \cdot x$ alineada con y (véase la Figura 3.9). La estimación de este algoritmo de $CED([x],[y])$, que será denotada como $BB(x, y)$, se realiza inicializando todos los nodos de partida con el valor 0 y calculando el valor mínimo de cualquier alineamiento óptimo que llegue a un nodo final. De este modo, el BB puede calcularse con una complejidad temporal $O(mn)$.

En [BB93] se muestra que, dadas dos cadenas x e y, el BB calcula $BB(x, y) = ED(x', y)$, donde x' es la subcadena de $x \cdot x$ que más se parece a y. La suboptimalidad de este método es debido a el hecho de que el alineamiento óptimo que encuentra podría empezar en $(i, 0)$ y finalizar en (i', n), con $i' \neq i + m$, mientras el camino que corresponde a $CED(x, y)$ debería verificar que $i' = i + m$. El camino correspondiente puede ser considerado como un seudo-alineamiento entre $[x]$ e $[y]$.

Podemos deducir entonces que el peso obtenido con el BB es una cota inferior de la CED, $BB(x, y) \leq CED([x],[y]])$.

3.4. Discusión

Parece tener sentido que, si queremos utilizar los contornos de las formas para el reconocimiento, la DEC nos puede ofrecer un buen método de comparación para las correspondientes cadenas cíclicas.

No obstante, el principal problema que se encuentra en las técnicas basadas en la distancia de edición, es el hecho de que los descriptores de contorno, que se utilizan para aproximar las formas, dependen en gran medida del ruido. Este ruido es el causante de las inconsistencias en la segmentación a las cuales la distancia de edición es muy sensible. Cada componente de una cadena o se tiene que alinear con una y sólo una componente de la otra cadena (siendo esta una sustitución) o se inserta o borra. Esto hace difícil el alineamiento de regiones similares representadas por un número diferente de componentes. Por otro lado, es complicado definir los pesos de las operaciones de edición cuando nos encontramos en entornos continuos.

En el caso de que tengamos estos problemas, las técnicas de comparación que veremos en el siguiente capítulo, el alineamiento temporal no lineal y los modelos ocultos de Markov resultarán más adecuadas.

Capítulo 4

Alineamiento temporal no lineal y modelos ocultos de Markov

Fio: «*What did he say?*»
Porco Rosso: «*He said "Cute kid. When did you start babysitting?"*»
Hayao Miyazaki, *Pòrco Rósso* (1998).

En este capítulo introduciremos dos técnicas que son conocidas desde hace tiempo por la comunidad científica dedicada al reconocimiento del habla, ya que ambas técnicas permiten modelar deformaciones elásticas, lo cual es útil para la comparación de la voz. Nos estamos refiriendo al alineamiento temporal no lineal y a los modelos ocultos de Markov. Las propiedades de estas dos técnicas también son interesantes en otras áreas, entre ellas, el modelado de contornos de formas bidimensionales.

4.1. Alineamiento temporal no lineal

El alineamiento temporal no lineal (ATNL) [SK83] (en inglés, *dynamic time warping*) es conocido por el área del reconocimiento de voz desde hace tiempo [Ita75, SK83, MRR80, RJ93, SC78]. Al tener que comparar dos pronunciaciones de voz descritas con cadenas de tramas, se tiene el problema de que estas, aun normalizando la duración, no tienen una correspondencia lineal. El ATNL permite una mejor medida de disimilitud cuando queremos modelar deformaciones elásticas, tal como ocurre en estas tramas de voz.

Debido a esta característica el ATNL se ha aplicado en otras disciplinas, como la bioinformática [AC01], las bases de datos [KR05], la biométrica [GD95], el reconocimiento de huellas digitales y firmas [KV00, MP99] , y en el reconocimiento de formas bidimensionales [AYV03, AO04, ARKF07], entre otras, como se ha mencionado en el Capítulo 2.

En la Figura 4.2 hay un ejemplo de alineamiento temporal lineal (ATL) [KR05] y de ATNL sobre dos funciones de curvatura que representan sendas formas. Nos ayudará a entender cuáles son las propiedades de este tipo de medida. Observando las cadenas nos damos cuenta de que las formas se parecen bastante, sin embargo, existe una serie de desfases temporales no uniformes. Como se puede ver el resultado con el ATL no tiene

demasiado sentido. El ATNL proporciona un alineamiento que parece más sensato para comparar este tipo de cadenas.

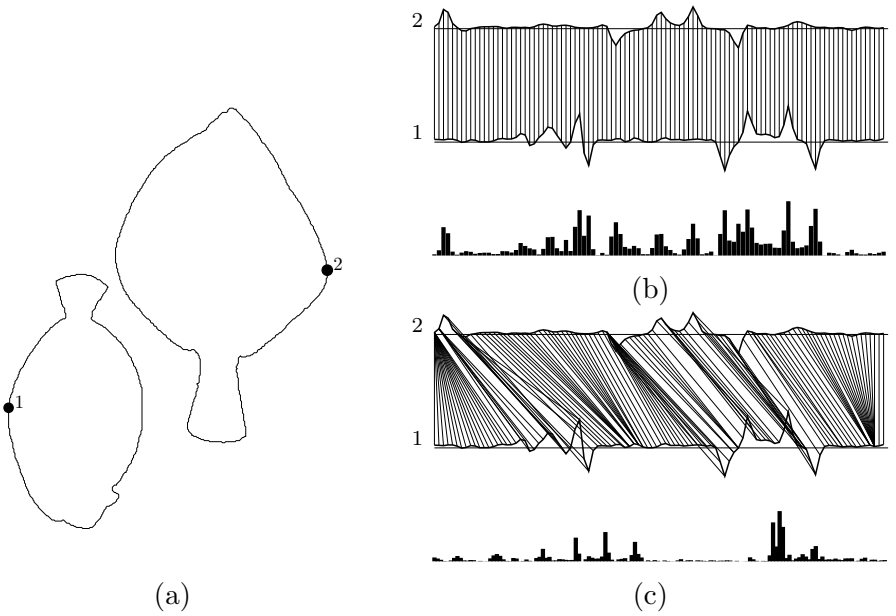


Figura 4.1: (a) Contornos de peces. Se indica en los contornos, mediante puntos, el inicio de las cadenas de valores de curvatura. Para realizar el alineamiento además hemos seleccionado el mismo número de puntos para ambas figuras de manera equidistante. (b) Alineamiento temporal lineal. El coste de los puntos alineados se muestra bajo el alineamiento. El coste del ATL es la suma de estos costes. (c) Alineamiento temporal no lineal.

De manera formal[1]. Sean $x = x_0x_1 \dots x_{m-1}$, donde $x_i \in \Sigma$, para $0 \leq i < m$ e $y = y_0y_1 \dots y_{n-1}$, donde $y_j \in \Sigma$, para $0 \leq j < n$. Un alineamiento entre x e y se define como una lista de pares (i_0, j_0), (i_1, j_1), $\dots$, (i_{k-1}, j_{k-1}) con las siguientes restricciones:

1. $0 \leq i_\ell < m$ y $0 \leq j_\ell < n$ para $0 \leq \ell < k$,
2. $0 \leq i_{\ell+1} - i_\ell \leq 1$ y $0 \leq j_{\ell+1} - j_\ell \leq 1$ para $0 \leq \ell < k$,
3. $(i_\ell, j_\ell) \neq (i_{\ell+1}, j_{\ell+1})$ para $0 \leq \ell < k$.

La segunda restricción proporciona dos propiedades importantes en el alineamiento: la monotonía, esto es, el camino de alineamiento no debería ir hacia atrás en el «tiempo»; y la continuidad, es decir, el camino de alineamiento no debería saltar en el «tiempo».

Significando el par (i_ℓ, j_ℓ), que x_{i_ℓ} está alineado con y_{j_ℓ}, el coste de un alineamiento es

$$\sum_{0 \leq \ell < k} \delta(x_{i_\ell}, y_{j_\ell}),$$

[1]Para facilitar la notación, al contrario que en las distancias de edición (véase el Capítulo 3), las cadenas empiezan con subíndice 0.

donde δ es la función local de disimilitud, la función que mide la distancia entre dos componentes de la cadena.

Un alineamiento óptimo es un alineamiento con coste mínimo. La Figura 4.2a muestra un alineamiento óptimo entre dos cadenas.

La medida de disimilitud basada en el alineamiento temporal no lineal $ATNL(x, y)$ se define como el peso de un alineamiento óptimo entre x e y, es decir, $ATNL(x, y) = d(m-1, n-1)$ [SK83], donde

$$d(i,j) = \begin{cases} \delta(x_0, y_0), & \text{si } i = j = 0; \\ d(i-1, j) + \delta(x_i, y_0), & \text{si } i > 0 \text{ y } j = 0; \\ d(i, j-1) + \delta(x_0, y_j), & \text{si } i = 0 \text{ y } j > 0; \\ \text{mín} \left\{ \begin{array}{l} d(i-1, j-1), \\ d(i-1, j), \\ d(i, j-1) \end{array} \right\} + \delta(x_i, y_j), & \text{en otro caso.} \end{cases} \tag{4.1}$$

Esta ecuación recursiva puede resolverse iterativamente con programación dinámica en tiempo $O(mn)$. El problema se reduce al cálculo de un camino óptimo en un grafo de alineamiento G_A, un grafo acíclico ponderado con $O(mn)$ arcos. El grafo de alineamiento es una malla formada por nodos (i, j), donde $0 \leq i < m$ y $0 \leq j < n$, conectados por arcos horizontales, verticales y diagonales, como puede verse en la Figura 4.2b. Todos los arcos que terminan tienen el mismo peso (o coste), $\delta(x_i, y_j)$. Todos los caminos de alineamiento empiezan en el nodo $(0, 0)$ y acaban en el nodo $(m-1, n-1)$.

Los alineamientos entre componentes de las cadenas, en el ATNL, pueden ser considerados como sustituciones en las distancias de edición, pero el ATNL permite correspondencias de uno a muchos. Esto hace que el ATNL sea apropiado para modelar distorsiones elásticas de cadenas que describan formas o series temporales. Por otro lado, los alineamientos que produce el ATNL no tienen ni inserciones ni borrados. Esto es preferible a las distancias de edición cuando estas no encajan de manera natural (véase la Sección 3.4).

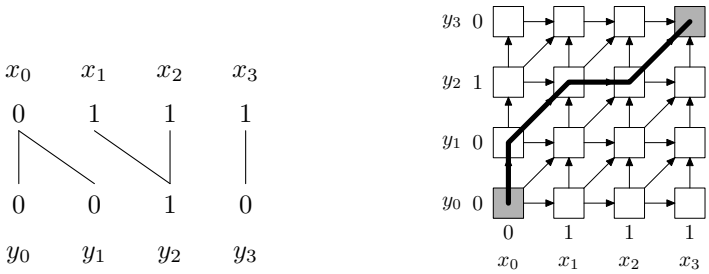


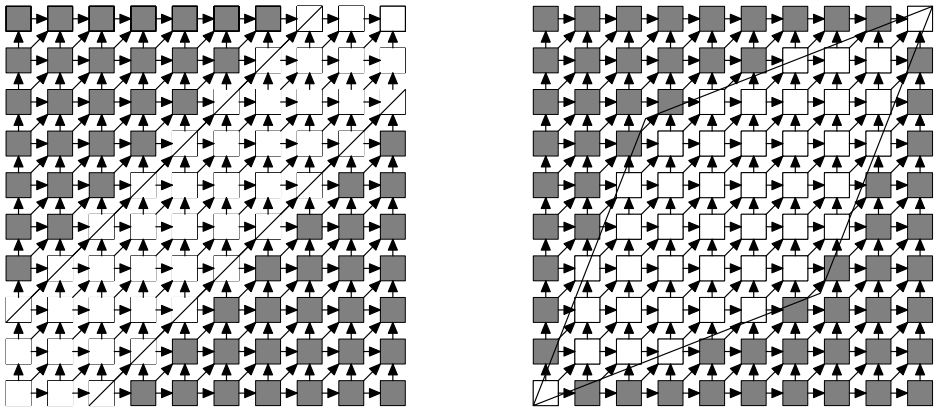
Figura 4.2: (a) Un alineamiento óptimo entre las cadenas discretas $x = 0111$ e $y = 0010$, donde $\delta(x_i, y_j) = |x_i - y_j|$. (b) Grafo de alineamiento para obtener la solución de la ecuación recursiva (4.1). La línea gruesa representa el alineamiento óptimo asociado a (a).

4.1.1. Restricciones globales

Además de las restricciones enumeradas anteriormente, al ATNL se le suelen imponer restricciones globales con el objetivo de que el camino del alineamiento no se aleje demasiado de la diagonal del grafo. Para conseguir esto podemos utilizar una banda de alineamiento, en la que el camino sólo puede visitar los nodos que están dentro de esta banda. De este modo, podemos añadir una cuarta restricción a las aparecidas anteriormente:

4. $|i_l - j_l| \leq s$ para $0 \leq \ell < k$.

Si $m = n$, en el caso de la banda de Sakoe-Chiba [SC78], s es independiente de i. Para el paralelogramo de Itakura [Ita75], s es función de i. En la Figura 4.3a, tenemos un ejemplo de la banda de Sakoe-Chiba, la más utilizada. También es conocida la ventana del paralelogramo de Itakura (Figura 4.3b). Estas bandas pueden aumentarse o reducirse de acuerdo a las necesidades.



(a) (b)

Figura 4.3: (a) Banda de Sakoe-Chiba. (b) Paralelogramo de Itakura. Los nodos en blanco son los que se pueden visitar para cada caso.

Existen dos razones principales para utilizar restricciones globales. Una de ellas es que agilizan el cálculo del ATNL. Calculando $ATNL_s$ (alineamiento temporal no lineal con una banda de Sakoe-Chiba s) conseguimos reducir su coste computacional a $O(\text{máx}(m, n)w)$. En segundo lugar, puede interesarnos que estos caminos extraños, llamados en la literatura patológicos, no se produzcan, ya que estos significan que una pequeña parte de una cadena se alinea con una parte grande de la otra y en ciertos campos de aplicación esto no es adecuado.

4.1.2. Restricciones locales

Otra manera de imponer restricciones al alineamiento es cambiar el conjunto de producciones [MRR80, RJ93]. La idea básica consiste en utilizar un conjunto de producciones

para tal fin (véase la Figura 4.4).

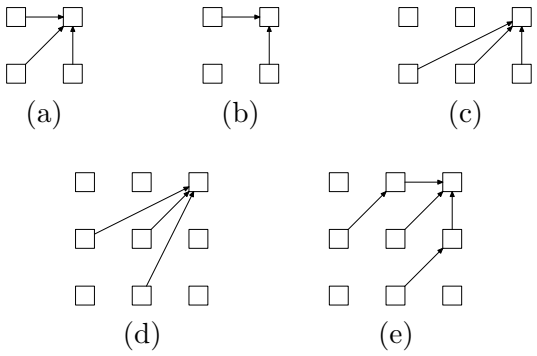


Figura 4.4: Diferentes conjuntos de producciones.

Las producciones permiten imponer restricciones de continuidad, simetría, monotonía, etc. Es posible también ponderar las producciones como se puede ver en la Figura 4.5.

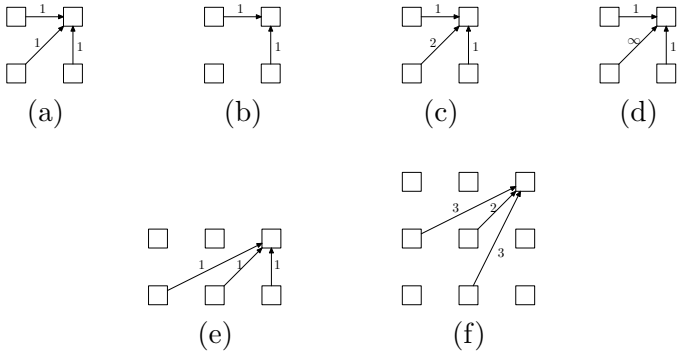


Figura 4.5: Conjuntos de producciones ponderadas.

Así pues, $ATNL(x, y) = d(m-1, n-1)$, donde la ecuación recursiva, de manera genérica, es

$$d(i,j) = \min_{(a,b)\in P}\{d(i-a, j-b) + w(a,b)\cdot\delta(x_i, y_j)\},$$

siendo w una función que devuelve el peso para las producciones dadas.

En particular, en las siguientes secciones sólo trabajaremos con las producciones de la Figura 4.4a para todas las ponderaciones posibles. Con este tipo de producciones, podemos simplificar la notación, utilizando $\gamma = (w(1,0), w(1,1), w(0,1))$, con lo que si $\gamma = (1,1,1)$ nos estamos refiriendo a las producciones ponderadas de la Figura 4.5a. La ecuación recursiva para $ATNL_\gamma(x, y) = d(m-1, n-1)$ finalmente queda como

$$d(i,j) = \begin{cases} \delta(x_0, y_0), & \text{si } i = j = 0; \\ d(i-1,j) + w(1,0) \cdot \delta(x_i, y_0), & \text{si } i > 0 \text{ y } j = 0; \\ d(i,j-1) + w(0,1) \cdot \delta(x_0, y_j), & \text{si } i = 0 \text{ y } j > 0; \\ \text{mín} \left\{ \begin{array}{l} d(i-1,j-1) + w(1,1) \cdot \delta(x_i, y_j), \\ d(i-1,j) + w(1,0) \cdot \delta(x_i, y_j), \\ d(i,j-1) + w(0,1) \cdot \delta(x_i, y_j) \end{array} \right\}, & \text{en otro caso.} \end{cases} \tag{4.2}$$

4.1.3. Normalización

Las distancias por el ATNL que estamos considerando pueden presentar un inconveniente: una propensión a dar valores altos para cadenas largas, y bajos para cadenas cortas, del mismo modo que la distancia de edición, como se comentó en la Sección 3.2.

Puede resultar necesario o conveniente efectuar alguna forma de normalización por la duración de las cadenas comparadas. Una posible definición de la distancia por el ATNL normalizado (ATNLN) sería:

$$ATNLN(x,y) = \min_{(i_0,j_0)\ldots(i_{k-1},j_{k-1})} \frac{\sum_{\ell=0}^{k} w(i_\ell - i_{\ell-1}, j_\ell - j_{\ell-1}) \cdot \delta(x_{i_\ell}, y_{j_\ell})}{\sum_{\ell=0}^{k} w(i_\ell - i_{\ell-1}, j_\ell - j_{\ell-1})}.$$

Para que esta ecuación sea resoluble con programación dinámica, con el mismo coste computacional que la distancia no normalizada, esta distancia debe presentar un denominador que no dependa del alineamiento. Ciertos conjuntos de producciones permiten trabajar con denominadores constantes, como los mostrados en la Figura 4.6.

Si el denominador depende del alineamiento tendremos que hacer uso de los métodos comentados en la Sección 3.2.

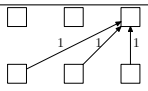
Producciones ponderadas	Denominador
	$n+m$
	$n+m$
	n

Figura 4.6: Conjuntos de producciones ponderadas que permiten la normalización con el mismo coste computacional.

4.2. Aceleración e invarianza al punto inicial con el ATNL

Como hemos mencionado anteriormente el ATNL es un algoritmo costoso computacionalmente, $O(mn)$. Es por esto que existe mucho trabajo orientado a paliar este problema, en especial en el campo de las bases de datos de series temporales [Keo02, ZS03, VHGK03, RK04, KPZ$^+$04, KR05, SMC05, VLM06, Lem09]. Aunque quizás el más conocido sea [Keo02] (y en el que los demás están basados), donde el autor obtiene una cota inferior (de coste lineal) del ATNL utilizando la ventana de Sakoe (véase la Sección 4.1.1) y la distancia de Minkowsky (Sección 2.3.1.1). Siendo x e y dos cadenas, y m el tamaño de ambas cadenas, la cota inferior se calcula de la siguiente manera:

$$LB\_Keogh(x,y) = \sum_{i=0}^{n-1} \begin{cases} (x_i - U_i)^2, & \text{si } q_i > U_i; \\ (x_i - L_i)^2, & \text{si } q_i < L_i; \\ 0, & \text{en otro caso,} \end{cases}$$

siendo U_i y L_i los límites superior e inferior de la cadena x para una banda de Sakoe dada.

En [Keo02] también se propone una técnica de indexación basada en [YJF98], en la que la serie temporal se aproxima con 16 componentes (realizando la media) que se transforman en 16 dimensiones a tratar por, por ejemplo, un *R*-tree* [Het].

Ambas técnicas parecen adecuadas en el caso de que nuestras cadenas (no cíclicas) tengan componentes de una dimensión, pero a medida que aumentemos las dimensiones de los elementos de las cadenas, el sistema se irá degenerando [WSB98].

Por otro lado, muy anteriormente en la literatura, también se ha utilizado AESA (*Approximating and Eliminating Search Algorithm*) [Vid86, Vid94] para acelerar el ATNL [VCR85, CVR87, VRCB88, VCBL88]. AESA tiene una fase de preproceso en la que se calculan todas las distancias entre los prototipos y se almacenan en una matriz D. En la fase de clasificación, en el inicio se elige arbitrariamente un candidato s como vecino más próximo (a la muestra a clasificar) y se calcula su distancia a la muestra d_s, eliminándolo del conjunto de prototipos y actualizando el vecino más próximo. A continuación esta distancia se utiliza para obtener una cota inferior G de la distancia de cada prototipo p a la muestra, utilizando para ello la propiedad de la desigualdad triangular (véase la Figura 4.7):

$$d(s,y) \leq d(s,s') + d(s',y) \Longrightarrow d(s',y) \geq d(s,y) - d(s,s')$$
$$d(s,s') \leq d(s,y) + d(s',y) \Longrightarrow d(s',y) \geq d(s,s') - d(s,y)$$

$$d(s',y) \geq |d(s,s') - d(s,y)|$$

Si la cota resulta ser superior a la distancia al vecino más cercano hasta el momento, el prototipo se poda. El siguiente prototipo a vecino más cercano se elige utilizando las cotas inferiores, y se repite todo el proceso (actualizando la cota en caso de que sea mayor que la obtenida hasta ese momento) hasta que no queden prototipos. El algoritmo AESA se muestra en la Figura 4.8.

AESA se caracteriza por una reducción drástica del cálculo de distancias. Es especialmente interesante cuando el cálculo de las distancias entre dos elementos es muy costoso,

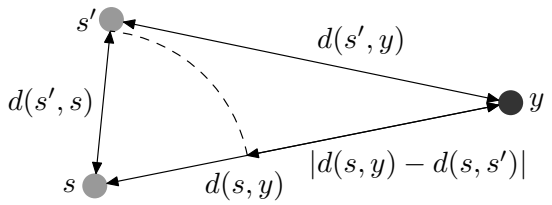


Figura 4.7: Cota inferior con la desigualdad triangular.

Figura 4.8: Algoritmo AESA.

Entrada: P: prototipos, x: muestra a clasificar
Salida: $nn \in P$: vecino más cercano
var $D \in \mathbb{R}^{|P| \times |P|}$: distancias entre prototipos
Inicio
  para $p \in P$ **hacer**
    $G[p] = 0$
  $nn =$ desconocido; $d_{nn} = \infty$
  $s =$ elemento cualquiera de P
  mientras $|P| > 0$ **hacer**
    $d_s = d(x, s)$
    $P = P - s$
    si $d_s < d_{nn}$ **entonces**
      $nn = s$; $d_{nn} = d_s$
    $siguiente =$ desconocido; $gmin = \infty$
    para $p \in P$ **hacer**
      $G[p] = \texttt{máx}(G[p], |D_B[s, p] - d_s|)$
      si $G[p] > d_{nn}$ **entonces**
        $P = P - p$
      si no
        si $G[p] < gmin$ **entonces**
          $gmin = G[p]$; $siguiente = p$
    $s = siguiente$
Fin

que es precisamente nuestro caso. Ahora bien, el ATNL no es una métrica, sin embargo, en [VCR85, CVR87, VRCB88, VCBL88] (mencionados anteriormente) se realizó un estudio con una tarea de reconocimiento de voz con palabras aisladas [RJ93] utilizando ATNL. En [CVR87], en 15 millones de tripletas no se encontró ningún caso en el que la desigualdad triangular no se cumpliera. Por tanto, pudieron aplicar AESA sin mayor problema. Incluso se permitieron el lujo de acelerar más el cálculo incrementando una holgura, H, para que la poda fuera mayor, hasta que la tasa de clasificación comenzaba a descender:

$$d(s', y) \geq |d(s, s') - d(s, y)| - H.$$

Esto en cuanto a cadenas ordinarias se refiere. En el caso de las cadenas cíclicas, el coste del ATNL es todavía mayor. Es bastante común, en la literatura relacionada con el reconocimiento de formas bidimensionales utilizando contornos [SKK03, AYV03, AO04, KWX$^+$06, ARKF07], pensar que el Lema 3.1, que cumple la distancia de edición, lo va a cumplir también el ATNL, pero esto no es así como veremos en el Capítulo 5. Por esto, tampoco podemos aplicar directamente el algoritmo divide y vencerás presentado en la Sección 3.3.2. En el Capítulo 5 veremos una adaptación de este algoritmo. En otros casos, incluso se comenta que es posible aplicar el algoritmo de ramificación y poda (véase la Sección 3.3.3), lo cual no parece posible según explicamos en la Sección 6.2.

En [KWX$^+$06], el mismo autor trata de llevar su método (comentado en párrafos anteriores) a las cadenas cíclicas (del mismo modo, sin tener en cuenta el incumplimiento del Lema 3.1). Pero, además de adolecer de los problemas anteriores (banda de Sakoe, Minkowsky, dimensionalidad) lo hace también utilizando un método de agrupación de cadenas en base a la similitud (trata todos los posibles desplazamientos cíclicos por separado) y por esto parece ser solamente utilizable con cadenas de componentes de una dimensión.

En el Capítulo 6 presentaremos mejores alternativas para la aceleración del reconocimiento con cadenas cíclicas de varias dimensiones y con cualquier distancia local. En particular, una adaptación del algoritmo presentado en la Sección 3.3.4 para acelerar el cálculo en vez de aproximar la distancia, un heurístico que acelera el reconocimiento en caso de que tengamos categorías etiquetadas y, por último, técnicas basadas en AESA, viendo cómo se soluciona, en parte, el problema del incumplimiento de la desigualdad triangular.

4.3. Modelos ocultos de Markov

Los modelos ocultos de Markov (MOMs) [Rab89] presentan propiedades similares al ATNL, pero además proporcionan un marco probabilístico para el entrenamiento y el reconocimiento. Debido a esto, de igual manera, se utilizaron en un principio en el reconocimiento del habla [Rab89, Lee89, RJ93], con mejores resultados que con el ATNL. Por supuesto, se utilizan también en otras disciplinas, como por ejemplo, entre muchas otras, procesamiento de señal [CNB98], bioinformática [DEKM98, HK96, HBT96], reconocimiento de gestos [EKR98, WB99], análisis y síntesis del comportamiento [JP98] o reconocimiento de texto manuscrito [PS00]. Hoy por hoy, los MOMs también están empezando a ser una alternativa en el reconocimiento de formas bidimensionales [HK91, FMJ97,

AYV00, CL01, BM04, BMF04, TGJ07].

Un MOM [Rab89] contiene un conjunto de estados, $\{S_1, S_2, \ldots, S_n\}$, cada uno con una distribución o densidad de probabilidad para la emisión de símbolos. En cada instante t, un estado produce un evento observable que sólo depende de ese estado. La transición de un estado a otro es un evento aleatorio que sólo depende del estado de partida.

Dado un alfabeto $\Sigma = \{v_1, v_2, \ldots, v_w\}$ (el conjunto de eventos observables es discreto y finito), un MOM discreto con n estados es una tripleta (A, B, π), donde:

- $A = \{a_{ij}\}$, para $1 \leq i, j \leq n$, es la matriz de probabilidades de transición (a_{ij} es la probabilidad de estar en el estado i en el instante t y estar en el estado j, en el instante $t + 1$);
- $B = \{b_{ij}\}$, para $1 \leq i, j \leq n$ y $1 \leq j \leq w$, es la matriz de probabilidades de observación (b_{ij}, o $b_i(v_j)$, es la probabilidad de observar v_j, estando en el estado i);
- $\pi = \{\pi_i\}$, para $1 \leq i \leq n$, es una distribución de probabilidad para los estados iniciales (π_i) es la probabilidad de estar en el estado i cuando $t = 1$.

Deben de satisfacerse, además, las siguientes condiciones, para todo i:

- $\sum_{1 \leq j \leq n} a_{ij} = 1$,
- $\sum_{1 \leq j \leq w} b_{ij} = 1$,
- $\sum_{1 \leq i \leq n} \pi_i = 1$.

La Figura 4.9a muestra una representación gráfica de un estado discreto y la Figura 4.9b muestra un modelo de Markov discreto completo.

Si los elementos observables de nuestras cadenas son continuos, la probabilidad puede venir dada por una distribución normal[2]:

$$b_i(v) = \mathcal{N}(v; \mu_i, \sigma_i) = \frac{1}{\sigma_i \sqrt{2\pi}} e^{-\frac{1}{2}(\frac{v-\mu_i}{\sigma_i})^2}.$$

Si los elementos de las cadenas, tienen varias dimensiones, se puede utilizar una distribución normal n-dimensional:

$$b_i(v) = \mathcal{N}(v; \mu_i, \Sigma_i) = \frac{1}{\sqrt{(2\pi)^n |\Sigma_i|}} e^{-\frac{1}{2}(v-\mu_i)' \Sigma_i^{-1} (v-\mu_i)}.$$

En el caso de que nuestros datos presenten distribuciones multimodales podremos hacer uso de mixturas de normales [Rab89].

[2]Utilizar una distribución normal es lo más habitual. Aunque se podrían utilizar otro tipo de distribuciones según el caso.

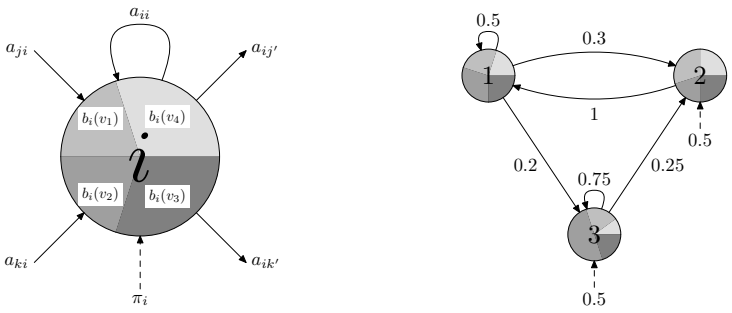


Figura 4.9: (a) Un estado de un MOM discreto que puede emitir cualquiera de los cuatro símbolos de acuerdo con el reparto de probabilidad ilustrado como un gráfico de tarta. (b) Una representación gráfica de un modelo de Markov discreto.

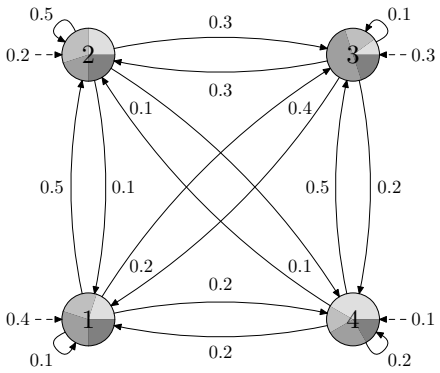
4.3.1. Topologías

El número de estados n y las probabilidades de transición entre estados que no son cero, $a_{ij} \neq 0$, definen la topología de los MOMs. La topología impone restricciones importantes sobre el proceso estocástico y produce distintos comportamientos en los modelos. Existen una gran cantidad de topologías dependiendo del dominio de aplicación. Pero quizás la clasificación más utilizada, considera simplemente dos tipos:

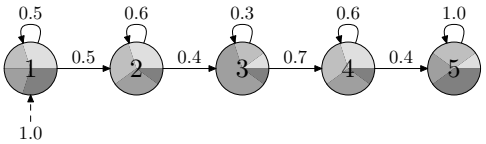
- Modelos ergódicos.
- Modelos izquierda-derecha (o temporales).

En el caso de los modelos ergódicos, quizás los más conocidos, la matriz de probabilidades de transición está completa y, por tanto, se pueden alcanzar todos los estados desde cualquier estado del modelo (véase la Figura 4.10a). Para algunas aplicaciones, en las que el reconocimiento requiere de una cierta temporalidad (como el reconocimiento de voz), otras topologías han dado mejores resultados que la topología ergódica. Nos estamos refiriendo a las topologías izquierda-derecha. En estas topologías, formalmente, ocurre lo siguiente que, $a_{ij} = 0$ para $j < i$. En las Figuras 4.10b y 4.10c se muestran ejemplos de dos casos particulares de estas topologías. Concretamente, en la Figura 4.10b tenemos un MOM izquierda-derecha lineal, donde desde un estado se transita a él mismo o al siguiente. Y en la Figura 4.10c se muestra una topología izquierda-derecha de Bakis.

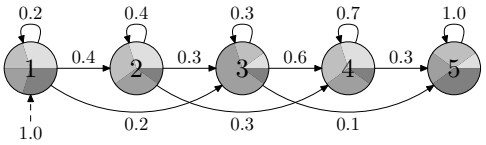
Aunque es habitual encontrarnos, en las definiciones de los modelos de Markov, con la distribución de probabilidad para los estados iniciales, π, en lo siguiente, utilizaremos en su lugar un estado S_0 no emisor y al que tampoco se puede transitar desde ningún estado, cuyas transiciones contendrán estas mismas probabilidades de π a los estados correspondientes. Con lo que el modelo de Markov quedará definido como $\lambda = (A, B)$. Así, con esta nueva forma de definir los modelos de Markov, por ejemplo, la Figura 4.9 tendría un estado no emisor S_0 del cual saldrían dos transiciones, una al estado S_2 con probabilidad 0.5 y otra al estado S_3 con probabilidad 0.5.



(a)



(b)



(c)

Figura 4.10: Ejemplos de topologías. (a) Topología ergódica. (b) Topología izquierda-derecha lineal. (c) Topología izquierda-derecha de Bakis.

4.3.2. Los tres problemas y uno más

Para un MOM, $\lambda = (A, B)$ y una cadena de elementos observados[3], $x = x_1 x_2 \dots x_m$, hay tres problemas básicos para que los modelos sean útiles en aplicaciones prácticas:

- El problema de la evaluación, es decir, la probabilidad de x, dado λ, $P(x|\lambda)$.
- El problema de la decodificación, es decir, obtener la secuencia de estados, $Q_0, \dots, Q_m$, que con mayor probabilidad produjo x (esta secuencia de estados puede verse también como un alineamiento óptimo entre x y λ).
- El problema del aprendizaje, es decir, estimar λ para maximizar la probabilidad de generar x.

Existe además un cuarto problema relacionado con el aprendizaje, el problema de la topología. Consiste en estimar la topología de λ para, junto con el aprendizaje, maximizar la probabilidad de generar x. Para este problema no existen métodos efectivos para estimar el número de estados o la topología del modelo. Normalmente estos son escogidos de manera heurística dependiendo de las características de la aplicación.

4.3.3. El problema de la evaluación y el de la decodificación

Existen algoritmos eficientes para resolver los dos primeros problemas. El procedimiento *forward-backward* [BE67, BS68] resuelve el problema de la evaluación. Para ello, utiliza dos variables, la variable *forward*, $\alpha_t(i)$ y, la variable *backward*, $\beta_t(i)$. La primera, $\alpha_t(i)$, se define como

$$\alpha_t(i) = P(x_1 \dots x_t, Q_t = S_i|\lambda)$$

y representa la probabilidad de haber observado la cadena $x_1 \dots x_t$ hasta el instante t, estando en el estado S_i. Se puede calcular recursivamente con lo siguiente:

$$\alpha_t(i) = \begin{cases} 1, & \text{si } t = 0 \text{ e } i = 0; \\ \left(\sum_{j=0}^{n} \alpha_{t-1}(j) a_{ji}\right) b_i(x_t), & \text{si } 1 \leq t \leq m \text{ y } 1 \leq i \leq n; \\ 0, & \text{en otro caso.} \end{cases}$$

La variable *backward* se define como

$$\beta_t(i) = P(x_{t+1} \dots x_m|Q_t = S_i, \lambda)$$

y representa la probabilidad de haber observado la cadena $x_{t+1} \dots x_T$, estando en el estado S_i en el instante t. Se puede calcular recursivamente con lo siguiente,

[3]Para facilitar la notación, en el caso de los modelos de Markov, las cadenas empiezan con subíndice 1.

$$\beta_t(i) = \begin{cases} 1, & \text{si } t = m \text{ y } 1 \leq i \leq n; \\ \sum_{j=1}^{n} a_{ij} b_j(x_{t+1}) \beta_{t+1}(j), & \text{si } t = m-1, \ldots, 1 \text{ y } 1 \leq i \leq n; \\ \sum_{j=1}^{n} a_{ij} \beta_{t+1}(j), & \text{si } t = 0 \text{ e } i = 0; \\ 0, & \text{en otro caso.} \end{cases}$$

$P(x|\lambda)$ se calcula como

$$P(x|\lambda) = \sum_{i=0}^{n} \alpha_t(i)\beta_t(i), \quad \forall t.$$

Fijando $t = m$ obtenemos

$$P(x|\lambda) = \sum_{i=0}^{n} \alpha_m(i).$$

El problema de la decodificación, como se ha comentado, se puede resolver también de manera eficiente utilizando el algoritmo de Viterbi [Vit67, For73]. Básicamente se utiliza una expresión recursiva similar a la del cálculo de la variable *forward*, pero con maximizaciones en vez de sumas:

$$\phi_t(i) = \begin{cases} 1, & \text{si } t = 0 \text{ e } i = 0; \\ \text{máx}_{0 \leq i \leq n} \left(\phi_{t-1}(j) a_{ji}\right) b_i(x_t), & \text{si } 1 \leq t \leq m \text{ y } 1 \leq i \leq n; \\ 0, & \text{en otro caso.} \end{cases}$$

El algoritmo de Viterbi devuelve la probabilidad de la secuencia de estados que más probablemente generó la cadena x en

$$\hat{P}(x|\lambda) = \max_{0 \leq i \leq n} \phi_m(i). \tag{4.3}$$

Para obtener la secuencia de estados más probable deberemos almacenar de dónde venimos (cuál ha sido el máximo), en cada paso recursivo [Rab89], con

$$\psi_t(i) = \arg \max_{0 \leq i \leq n} \left(\phi_{t-1}(j) a_{ji}\right),$$

para al final obtener el camino, es decir, la secuencia óptima de estados, $\hat{Q} = \hat{Q}_m, \hat{Q}_{m-1}, \ldots, \hat{Q}_0$, utilizando retroceso:

$$\hat{Q}_m = \arg \max_{0 \leq i \leq n} \left(\phi_m(i)\right),$$

$$\hat{Q}_t = \psi_{t+1}(\hat{Q}_{t+1}), \text{para } t = m-1, m-2, \ldots, 0.$$

Además de resolver el problema de la decodificación, el algoritmo de Viterbi es utilizado, frecuentemente, para resolver también el problema de la evaluación. Esta puntuación (4.3) es una muy buena aproximación a la probabilidad que devuelve el *forward*[4].

Las tres ecuaciones recursivas pueden ser resueltas con una versión iterativa utilizando programación dinámica en tiempo $O(n^2m)$ [RJ93].

4.3.4. El problema del aprendizaje

No existe un algoritmo que resuelva de manera óptima el problema del entrenamiento. Aunque sí existen procedimientos eficientes, como, el algoritmo de reestimación Baum-Welch [BE67, BS68, BPSW70, Bau72] que mejora iterativamente la estimación de los parámetros hasta que se alcanza un máximo local. Este algoritmo es una particularización del conocido algoritmo *expectation-maximization* (EM) [DLR77, Wu83].

Para describir el algoritmo que realiza la reestimación, se deben introducir dos variables:

- $\xi_t(i,j)$: representa la probabilidad de pasar del estado S_i en el instante t al estado S_j en el instante $t+1$, dada la cadena x y el modelo λ, es decir,

$$\xi_t(i,j) = P(Q_t = S_i, Q_{t+1} = S_j | x, \lambda).$$

  Esta variable se calcula utilizando las variables *forward* y *backward*:

$$\begin{aligned}\xi_t(i,j) &= P(Q_t = S_i, Q_{t+1} = S_j | x, \lambda)) \\ &= \frac{P(Q_t = S_i, Q_{t+1} = S_j, x|\lambda)}{P(x|\lambda)} \\ &= \frac{\alpha_t(i) a_{ij} b_j(x_{t+1}) \beta_{t+1}(j)}{P(x|\lambda)} \\ &= \frac{\alpha_t(i) a_{ij} b_j(x_{t+1}) \beta_{t+1}(j)}{\sum_i \sum_j \alpha_t(i) a_{ij} b_j(x_{t+1}) \beta_{t+1}(j)}.\end{aligned}$$

- $\gamma_t(i)$: representa la probabilidad de estar en el estado S_i en el instante t, dada la cadena x y el modelo λ, es decir,

$$\gamma_t(i) = P(Q_t = S_i | x, \lambda).$$

  Esta se calcula con

[4] Ofreciendo ciertas ventajas de implementación y de costes respecto a este último.

$$\begin{aligned}\gamma_t(i) &= P(Q_t = S_i|x,\lambda)\\ &= \frac{P(Q_t = S_i, x|\lambda)}{P(x|\lambda)}\\ &= \frac{\alpha_t(i)\beta_t(i)}{\sum_i \alpha_t(i)\beta_t(i)}.\end{aligned}$$

Pero también puede calcularse en términos de la variable $\xi_t(i,j)$:

$$\gamma_t(i) = \sum_{j=1}^{n} \xi_t(i,j).$$

Dadas estas dos variables, el proceso de reestimación determina los parámetros de λ de la siguiente manera:

$$\begin{aligned}\bar{a}_{ij} &= \frac{\text{Nº esperado de veces que se pasa de } S_i \text{ a } S_j}{\text{Nº esperado de veces que se pasa por } S_i}\\ &= \frac{\sum_{t=0}^{m-1} \xi_t(i,j)}{\sum_{t=0}^{m-1} \gamma_t(i)},\end{aligned}$$

$$\begin{aligned}\bar{b}_i(v_j) &= \frac{\text{Nº esperado de veces que se pasa por } S_i \text{ y se observa } v_j}{\text{número esperado de veces que se pasa por } S_i}\\ &= \frac{\sum_{\substack{t=1\\ \forall x_t = v_j}}^{m-1} \gamma_t(i)}{\sum_{t=1}^{m-1} \gamma_t(i)}.\end{aligned}$$

Con lo que podremos reestimar el nuevo modelo, $\bar{\lambda} = (\bar{A}, \bar{B})$, a partir del anterior $\lambda = (A, B)$. Baum *et al.* demostraron que

$$P(x|\bar{\lambda}) \geq P(x|\lambda),$$

lo cual significa que, en cada iteración la verosimilitud se incrementa, hasta que se converge a un máximo. El resultado final de este proceso es un MOM entrenado utilizando estimación por máxima verosimilitud. El máximo que se alcanza es local, con lo que la inicialización, de los parámetros del modelo, cobra especial relevancia.

Por otro lado, una sola cadena no suele ser suficiente para entrenar un MOM [Rab89]. Debido a que estas fórmulas están basadas en el cálculo de frecuencias de los eventos, las fórmulas de reestimación para múltiples cadenas se modifican sumando todas las frecuencias individuales de cada cadena, asumiendo que las cadenas son independientes,

$$P(X|\lambda) = \prod_{l=1}^{L} P(x^{(l)}|\lambda),$$

siendo X, el conjunto de cadenas, $X = \{x^{(1)}, x^{(2)}, \ldots, x^{(L)}\}$, las fórmulas de reestimación quedan de la siguiente manera:

$$\bar{a}_{ij} = \frac{\sum_{l=1}^{L} \frac{1}{P(x^{(l)}|\lambda)} \sum_{t=0}^{m^{(l)}-1} \alpha_t^l(i) a_{ij} b_j(x_{t+1}^{(l)}) \beta_{t+1}^l(j)}{\sum_{l=1}^{L} \frac{1}{P(x^{(l)}|\lambda)} \sum_{t=0}^{m^{(l)}-1} \alpha_t^l(i) \beta_{t+1}^l(j)}, \tag{4.4}$$

$$\bar{b}_i(v_j) = \frac{\sum_{l=1}^{L} \frac{1}{P(x^{(l)}|\lambda)} \sum_{\substack{t=1 \\ \forall x_t^{(l)}=v_j}}^{m^{(l)}-1} \alpha_t^l(i) \beta_t^l(i)}{\sum_{l=1}^{L} \frac{1}{P(x^{(l)}|\lambda)} \sum_{t=1}^{m^{(l)}-1} \alpha_t^l(i) \beta_t^l(i)}. \tag{4.5}$$

Hasta aquí hemos comentado el algoritmo Baum-Welch para entrenar MOMs con cadenas de valores discretos. En el caso de que queramos entrenar nuestro modelo con cadenas de valores continuos y/o con elementos de varias dimensiones, la extensión de las fórmulas de reestimación, a distribuciones normales, distribuciones normales n-dimensionales y/o mixturas, es inmediata [Rab89].

En la literatura existe también un algoritmo de entrenamiento más sencillo, el *segmental K-means* [JR90]. Su sencillez radica en el hecho de que en las fórmulas de reestimación (para distinguirlas de las del algoritmo de Baum-Welch las llamaremos $\hat{a}_{ij}$ y $\hat{b}_i(v_j)$) sólo se utilizan las frecuencias de las secuencias de estados óptimas, que nos da el algoritmo de Viterbi ($\hat{Q} = \hat{Q}_m, \hat{Q}_{m-1}, \ldots, \hat{Q}_0$) y no frecuencias basadas en el número esperado. En muchas ocasiones, este método y el de Baum-Welch ofrecen resultados comparables.

4.4. Invarianza al punto inicial con los MOMs

Si nos fijamos en cómo se consigue la invarianza al punto inicial con los MOMs, veremos que en la literatura aparece: la elección de rotación de referencia [HK91] (véase la Sección 2.4.1), características invariantes a la rotación [CL01] (Sección 2.4.2), una topología circular [AYV00] y utilizar, sin más, un modelo ergódico [BM04, TGJ07], donde el entrenamiento resolverá el problema.

[HK91, BM04, TGJ07] emplean topologías ergódicas, lo cual tiene como consecuencia que los estados se utilizan para explicar múltiples observaciones a lo largo del contorno. Esto hace que el reconocimiento sea un problema complejo. Para reconocer cadenas parecen más adecuadas las topologías izquierda-derecha. Aunque típicamente contengan más estados, el número de transiciones es escaso, con lo que la complejidad de los algoritmos se reduce. Ahora bien, las cadenas cíclicas no tienen un punto inicial (ni tampoco final), los MOMs izquierda-derecha parecen, en principio, inapropiados para modelarlas.

En [AYV00], los autores proponen una topología circular para modelar las cadenas cíclicas. La estructura propuesta elimina la necesidad de definir un punto de inicio: la cadena cíclica puede segmentarse para asociar estados consecutivos a segmentos consecutivos en las cadenas, pero no se hace ninguna asunción sobre cuál es el primer o último segmento. No obstante, como veremos en el Capítulo 7, la topología propuesta tiene los mismos problemas que la ergódica.

Como se ha comentado en capítulos anteriores la única solución para conseguir la invarianza al punto inicial es utilizar todo posible punto de inicio de las cadenas, es decir, utilizar cadenas cíclicas. Sin embargo, tal como hemos visto en la sección anterior los modelos de Markov sólo pueden generar cadenas ordinarias (no cíclicas).

En el Capítulo 7 propondremos extensiones para las cadenas cíclicas, al algoritmo de decodificación de Viterbi y a los algoritmos de entrenamiento *segmental K-means* y Baum-Welch. Por otro lado, veremos que el heurístico que aparece en la Sección 6.1, puede también acelerar el entrenamiento y la evaluación/decodificación de cadenas cíclicas.

4.5. Discusión

Con esto, acaba ya la parte del estado del arte. En este capítulo hemos concretado ya la problemática que se pretende resolver en la presente tesis y sobre qué técnicas.

Empieza ya la parte de las aportaciones.

Parte II

Aportaciones

Capítulo 5

Alineamiento temporal no lineal cíclico

Curtis: «*If you run away, I'll tell everyone you're chicken!*»
Porco Rosso: «*Chicken, pig, what's the difference?*»
Hayao Miyazaki, *Pòrco Rósso* (1998).

Presentaremos en este capítulo la extensión del ATNL a cadenas cíclicas, es decir, el ATNL cíclico. Veremos también un algoritmo eficiente para su cálculo y extensiones para otras ponderaciones, restricciones globales y normalización.

5.1. Introducción

Cuando dos formas representadas con cadenas tienen puntos de comienzo definidos, el ATNL proporciona una buena medida de disimilitud. Sin embargo, si queremos conseguir invarianza al punto inicial debemos considerar el marco de las cadenas cíclicas. En la Figura 5.1 hay un ejemplo en el cual se puede ver que se consigue un mejor alineamiento cuando se tiene el punto de inicio «equivalente» para las dos formas.

Tal como se dijo en la Sección 3.3. Un desplazamiento cíclico ρ de una cadena $x = x_0x_1 \dots x_{m-1}$ se define como $\rho(x_0x_1 \dots x_{m-1}) = x_1 \dots x_{m-1}x_0$. Siendo ρ^k la composición de k desplazamientos cíclicos y ρ^0 la identidad.

El Lema 3.1 nos dice que $DEC([x],[y]) = \text{mín}_{0\leq k<m}\, DE(\rho^k(x), y)$. Este lema es fundamental para poder utilizar el algoritmo divide y vencerás, ya que el tener sólo que rotar una cadena nos lleva a poder utilizar el grafo extendido y la propiedad de no cruce de caminos (Sección 3.3.2). Aunque a priori, pueda parecer que este lema se cumple también con el ATNL, esto no es así, como veremos a continuación.

Un alineamiento cíclico entre $[x]$ e $[y]$ es una lista de pares $(i_0, j_0), (i_1, j_1), \dots, (i_{k-1}, j_{k-1})$ tal que, para $0 \leq \ell < k$:

1. $0 \leq i_\ell < m$ and $0 \leq j_\ell < n$,
2. $0 \leq (i_{(\ell+1) \text{ mód } k} - i_\ell) \text{ mód } m \leq 1$ and $0 \leq (j_{(\ell+1) \text{ mód } k} - j_\ell) \text{ mód } n \leq 1$,
3. $(i_\ell, j_\ell) \neq (i_{(\ell+1) \text{ mód } k}, j_{(\ell+1) \text{ mód } k})$.

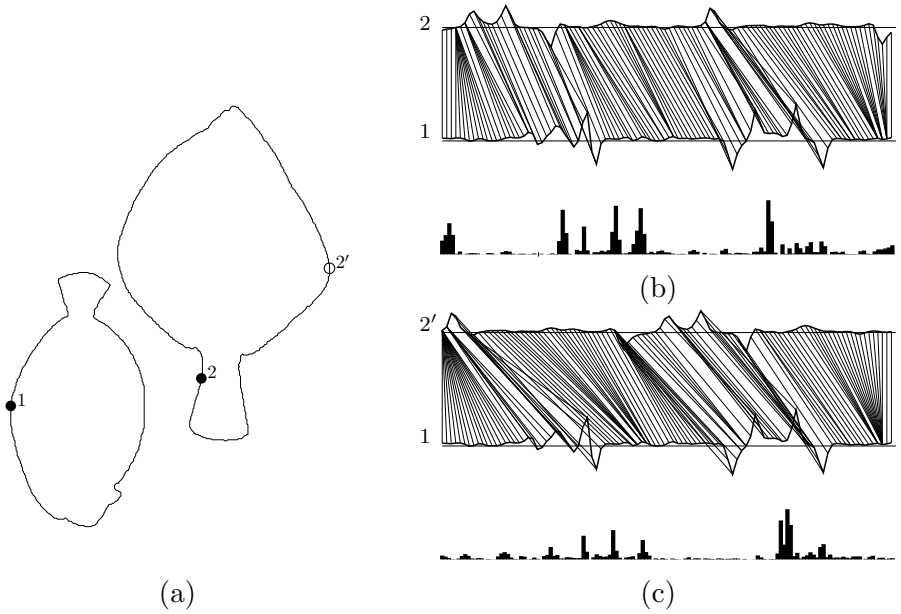


Figura 5.1: (a) Contornos de peces. Se indica en los contornos, mediante puntos, el inicio de las cadenas de valores de curvatura. Para realizar el alineamiento además hemos seleccionado el mismo número de puntos para ambas figuras. (b) Alineamiento óptimo si la cadena empieza en el punto 2. El coste de alinear cada par de puntos se muestra bajo el alineamiento. El coste del ATNL es la suma de estos costes. (c) Alineamiento óptimo si la cadena empieza en el punto $2'$.

El peso de un alineamiento cíclico (i_0, j_0), (i_1, j_1), ..., (i_{k-1}, j_{k-1}) se define como $\sum_{0 \le \ell < k} \delta(x_{i_\ell}, y_{j_\ell})$, donde δ es una distancia local de disimilitud. Un alineamiento cíclico óptimo es un alineamiento de coste mínimo.

La medida de disimilitud del alineamiento temporal no lineal cíclico $ATNLC([x], [y])$ se define como el coste del alineamiento óptimo entre $[x]$ e $[y]$.

A continuación mostraremos cómo puede ser definido el alineamiento cíclico óptimo en términos de alineamientos entre cadenas no cíclicas, es decir, utilizando la fuerza bruta en términos del ATNL. Definiremos en primer lugar la fuerza bruta para la ponderación $\gamma = (1, 1, 1)$, por ser esta la más utilizada y que conduce a un desarrollo más sencillo, y después, la fuerza bruta para cualquier ponderación (para el mismo conjunto de producciones). Finalmente, veremos cómo, utilizando una estrategia divide y vencerás, podemos acelerar el cálculo del ATNLC.

5.2. Ponderación $\gamma = (1, 1, 1)$

Empezamos por demostrar un lema acerca de la estructura de los alineamientos cíclicos óptimos.

Lema 5.1 *Si $m, n > 1$ e (i_0, j_0), (i_1, j_1), ..., (i_{k-1}, j_{k-1}) es un alineamiento óptimo entre dos cadenas $x_0 x_1 \dots x_{m-1}$ e $y_0 y_1 \dots y_{n-1}$, existe al menos un ℓ tal que $i_\ell \neq i_{(\ell+1) \bmod k}$ y $j_\ell \neq j_{(\ell+1) \bmod k}$.*

Demostración: Cualquier alineamiento que incluya (i_ℓ, j_ℓ), $(i_\ell+1, j_\ell)$, e $(i_\ell+1, j_\ell+1)$ puede ser «mejorado» quitando $(i_\ell + 1, j_\ell)$, ya que $\delta(x_{i_\ell+1}, y_{j_\ell}) \ge 0$. Análogamente, cualquier alineamiento que incluya (i_ℓ, j_ℓ), $(i_\ell, j_\ell+1)$, e $(i_\ell+1, j_\ell+1)$ puede ser «mejorado» quitando $(i_\ell, j_\ell + 1)$. □

Nos apoyamos en este lema para demostrar que el alineamiento óptimo puede expresarse en términos de alineamientos entre cadenas no cíclicas.

Lema 5.2 *La disimilitud ATNLC entre una cadena cíclica $[x]$ y una cadena cíclica $[y]$, $ATNLC([x], [y])$, puede ser calculada con*

$$ATNLC([x], [y]) = \min_{0 \le k < m} \min_{0 \le \ell < n} ATNL(\rho^k(x), \rho^\ell(y)).$$

Demostración: Trivial cuando $m = 1$ o $n = 1$. Consideremos que $m, n > 1$ y sea (i_0, j_0), (i_1, j_1), ..., (i_{k-1}, j_{k-1}) un alineamiento óptimo. Sea ℓ un índice tal que $i_\ell \neq i_{(\ell+1) \bmod k}$ y $j_\ell \neq j_{(\ell+1) \bmod k}$ (por el Lema 5.1). El coste de este alineamiento cíclico es $ATNL(\rho^{(i_\ell+1) \bmod k}(x), \rho^{(j_\ell+1) \bmod k}(y))$, que se considera en la doble minimización. □

De acuerdo con el Lema 5.2, el valor de $ATNLC([x], [y])$ puede ser obtenido de manera trivial en tiempo $O(m^2 n^2)$ considerando todos los desplazamientos cíclicos de ambas cadenas, resolviendo mn ecuaciones recursivas como la mostrada en (4.1).

Pero, ¿es posible realizar sólo desplazamientos cíclicos sobre una cadena para calcular el ATNLC (tal como se hace con la DEC)? Es decir, ¿podemos afirmar que se cumple la siguiente igualdad?

$$\min_{0\le k<m}\min_{0\le \ell<n} ATNL(\rho^k(x),\rho^\ell(y)) = \min_{0\le k<m} ATNL(\rho^k(x),y).$$

Al contrario de lo que se afirma en [SKK03, AYV03, AO04, KWX+06, ARKF07], la respuesta es no. En general, $ATNLC([x],[y])$ no es igual a $\min_{0\le k<m} ATNL(\rho^k(x),y)$ ni a $\min_{0\le l<n} ATNL(x,\rho^l(y))$ como muestra el siguiente contraejemplo. Sea $\delta(x_i,y_j)=|x_i-y_j|$; el valor de $ATNLC([101],[010])$ es 0, ya que $ATNL(110,100)=0$, pero,

- $ATNL(101,010)=3$, y,
- $ATNL(011,010)=ATNL(110,010)=ATNL(101,100)=ATNL(101,001)=1$.

Por lo tanto, no se puede extender directamente el algoritmo divide y vencerás [Mae90] al cálculo del ATNLC.

El siguiente teorema muestra una relación de igualdad que podemos explotar en un algoritmo eficiente:

Teorema 5.3 *La disimilitud ATNLC entre una cadena cíclica* $[x]$ *y una cadena cíclica* $[y]$ *es*

$$ATNLC([x],[y]) = \min_{0\le k<m}\Big(\min(ATNL(\rho^k(x),y), ATNL(\rho^k(x)x_k,y))\Big), \qquad (5.1)$$

donde $\rho^k(x)x_k$ es la concatenación de x_k a $\rho^k(x)$.
Demostración: Cada alineamiento produce una segmentación en x y una segmentación en y: todos los valores en un segmento están alineados con el mismo valor de la otra cadena cíclica (Lemma 5.1). Existe un problema cuando $y_{n-p-1}y_{n-p}\dots y_{n-1}$ e $y_0y_1\dots y_q$, para algún p, $q\ge 0$, debería pertenecer al mismo segmento de y, es decir, no hemos cortado el segmento limpiamente, al seleccionar y como desplazamiento cíclico de $[y]$ para usar el ATNL. En ese caso, el camino óptimo no se puede obtener simplemente realizando el desplazamiento sobre x, ya que y_{n-1} debe ser alineado con el último valor de $\rho^k(x)$ e y_0 debe ser alineado con el primer valor de $\rho^k(x)$, es decir, no pueden pertenecer al mismo segmento. Sin embargo, la cadena $\rho^k(x)x_k$ permite alinear $y_{n-p}y_{n-p+1}\dots y_n$ y $y_0y_1\dots y_q$ con el primer valor de $\rho^k(x)$, ya que x_k aparece también al final de $\rho^k(x)x_k$. Esta es la razón de que debamos considerar el segundo elemento de la minimización interior en (5.1). □

Corolario 5.4 *$ATNL(\rho^k(x),y)$, para cada valor de k, puede ser obtenido como el subproducto del cálculo de $ATNL(\rho^k(x)x_k,y)$. Por lo que debe añadirse una columna al grafo extendido para que incluya los caminos propios de este alineamiento especial.*

Demostración: El grafo de alineamiento que corresponde a $ATNL(\rho^k(x),y)$ es un subgrafo del grafo de alineamiento que corresponde a $ATNL(\rho^0(x)x_0,y)$. El camino óptimo en $ATNL(\rho^k(x),y)$ es un camino en el grafo $ATNL(\rho^k(x)x_k,y)$. □

En la Figura 5.2 se puede ver este hecho. Además se muestra que el contraejemplo ya no es tal.

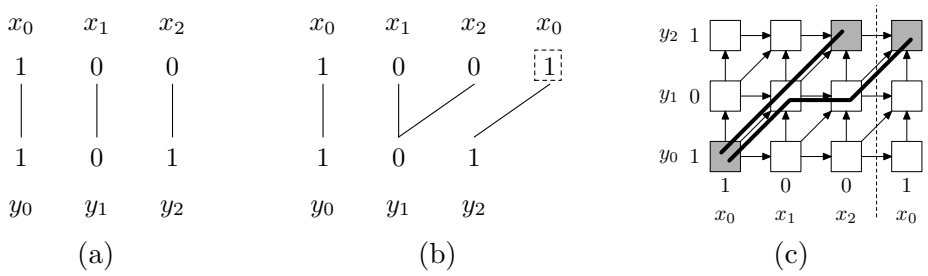


Figura 5.2: Contraejemplo mencionado anteriormente para el desplazamiento cíclico donde se obtiene el mínimo coste con la solución propuesta.

5.3. Cualquier ponderación

En muchos casos la ponderación $\gamma = (1, 1, 1)$ resulta apropiada para realizar comparaciones. No obstante, una ponderación diferente para este conjunto de producciones (Figura 4.4a) resulta conveniente en otros casos. Por ejemplo (como se ha comentado anteriormente), para obtener distancias normalizadas, con el mismo coste computacional, las ponderaciones deben de proporcionar un denominador que no dependa del alineamiento.

Como se explica en la demostración del Teorema 5.3, con la ponderación $\gamma = (1, 1, 1)$ sólo existen dos posibles cortes en un alineamiento, o hemos cortado uno de los segmentos de y, al elegir el desplazamiento cíclico, o no lo hemos hecho y la cadena se ha cortado limpiamente. En el caso de ponderaciones arbitrarias, además de estos dos tipos de cortes, se añaden otros dos. En la Figura 5.3 podemos ver el alineamiento de las cadenas que aparecen también en la Figura 4.2, pero esta vez para $\gamma = (1, 2, 1)$. Como se puede ver en la Figura 5.3a existe una pequeña diferencia, que es el alineamiento entre x_3 e y_2 y que en el grafo (véase la Figura 5.3b) corresponde a un ángulo recto en el camino de alineamiento.

El Lema 5.1 no se cumple (observa ahora que usamos pesos diferentes de $\gamma = (1, 1, 1)$), ya que en este caso, el alineamiento incluye $(2, 2)$, $(3, 2)$ y $(3, 3)$, pero no podemos quitar $(3, 2)$ para obtener uno de coste menor. Si cortamos y entre y_2 e y_3 se produce uno de estos dos nuevos tipos de corte a los que llamaremos N o И (por el parecido con la letra mayúscula y su reflejo). En el alineamiento $(2, 2)$, $(3, 2)$ y $(3, 3)$ se produce una ene reflejada, И, como se puede ver en la Figura 5.3a. El corte N se hubiera producido si el alineamiento hubiera incluido $(2, 2)$, $(2, 3)$ y $(3, 3)$ (el otro ángulo recto, en el grafo de alineamiento, Figura 5.3b).

Estos nuevos cortes invalidan también el Lema 5.2 para pesos arbitrarios de las producciones, ya que realizar todos los posibles desplazamientos cíclicos de las dos cadenas no es suficiente, como se muestra en el siguiente contraejemplo. Supongamos que queremos calcular $ATNL_\gamma([02], [131])$, siendo $\delta(x_i, y_j) = |x_i - y_j|$ y $\gamma = (1, 3, 1)$. Si desplazamos cíclicamente las dos cadenas obtenemos:

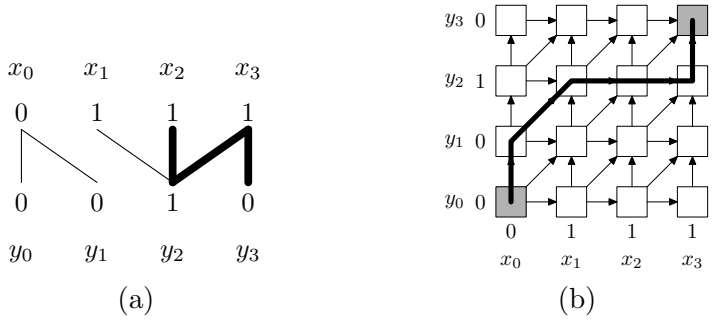


Figura 5.3: (a) Un alineamiento óptimo entre las cadenas discretas $x = 0111$ e $y = 0010$, donde $\delta(x_i, y_j) = |x_i - y_j|$ y $\gamma = (1, 2, 1)$. Aparece una ene reflejada que está marcada con trazo grueso. (b) Grafo de alineamiento. La línea gruesa representa el alineamiento óptimo asociado a (a).

- $ATNL_\gamma(02, 113) = ATNL_\gamma(02, 131) = ATNL_\gamma(20, 131) = ATNL_\gamma(20, 311) = 4$, y,
- $ATNL_\gamma(02, 311) = ATNL_\gamma(20, 113) = 6$.

Pero el resultado correcto para $ATNL_\gamma([02], [131])$ es 5 como se muestra en la Figura 5.4. En la Figura 5.4a se ilustra el alineamiento óptimo de $ATNL_\gamma(02, 131) = 4$ que supuestamente es el resultado del alineamiento cíclico, sin embargo, esto no es así ya que no tenemos en cuenta las ponderaciones en (4.2) para $i = 0$ y $j = 0$. x_0 debería de aparecer alineado con y_2 tal como ocurre en la Figura 5.4b.

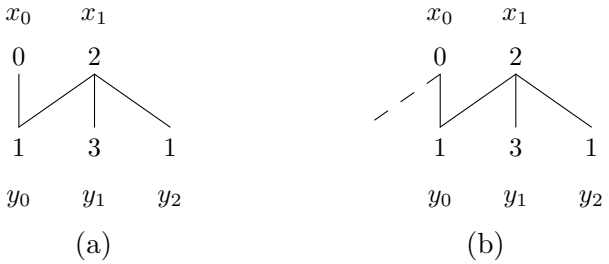


Figura 5.4: (a) Alineamiento óptimo entre las cadenas $x = 02$ e $y = 131$, donde $\delta(x_i, y_j) = |x_i - y_j|$ y $\gamma = (1, 3, 1)$. (b) Alineamiento óptimo entre las cadenas cíclicas $x = [02]$ e $y = [131]$. La línea punteada indica el alineamiento que falta en (a) para que este sea una alineamiento cíclico correcto.

Ahora bien, para conseguir este resultado debemos tener en cuenta cómo salimos del grafo por el nodo $(m-1, n-1)$ para entrar en el nodo $(0, 0)$, es decir, necesitamos saber con qué ponderación hemos salido. Esto nos lleva a encontrar otra expresión para el ATNL, a esta nueva expresión la llamaremos ATNL2 debido a que nos va a servir para expresar la fuerza bruta con el desplazamiento cíclico de las 2 cadenas.

Lema 5.5 *Sea* $ATNL2_\gamma(x,y) = d(m,n)$ *y la ecuación recursiva para su resolución:*

$$d(i,j) = \begin{cases} 0, & \text{si } i = j = 0; \\ d(i-1,j) + w(1,0)\cdot\delta(x_i,y_0), & \text{si } i > 0 \text{ y } j = 0; \\ d(i,j-1) + w(0,1)\cdot\delta(x_0,y_j), & \text{si } i = 0 \text{ y } j > 0; \\ d(i-1,j-1) + & \\ \text{mín}\left\{ \begin{array}{l} w(1,1)\cdot\delta(x_0,y_0), \\ w(0,1)\cdot\delta(x_0,y_{m-1}) + w(1,0)\cdot\delta(x_0,y_0) \end{array} \right\}, & \text{si } i = m \text{ y } j = n; \\ \text{mín}\left\{ \begin{array}{l} d(i-1,j-1) + w(1,1)\cdot\delta(x_i,y_j), \\ d(i-1,j) + w(1,0)\cdot\delta(x_i,y_j), \\ d(i,j-1) + w(0,1)\cdot\delta(x_i,y_j) \end{array} \right\}, & \text{en otro caso.} \end{cases} \tag{5.2}$$

La disimilitud ATNLC para ponderaciones arbitrarias entre una cadena cíclica $[x]$ *de longitud* m *y una cadena cíclica* $[y]$ *de longitud* n, $ATNLC_\gamma([x],[y])$, *puede ser calculada con*

$$ATNLC_\gamma([x],[y]) = \min_{0\le k<m}\left(\min_{0\le \ell<n} ATNL2_\gamma(\rho^k(x),\rho^\ell(y))\right).$$

Demostración: Al poderse producir un corte en N o en $\not N$, cuando se selecciona un desplazamiento cíclico, debemos tener en cuenta cómo hemos salido del grafo y entrado en $(0,0)$. Esto se consigue con las diferencias de (5.2) con respecto a (4.2): (i) el caso en el que $i = m$ y $j = n$, con el que conseguimos conocer el peso adecuado para entrar en $(0,0)$; y (ii) el caso $i = j = 0$ que tiene como resultado 0, ya que no necesitamos añadir aquí la distancia local porque ya la habremos añadido con el caso (i). No obstante, en el caso (i) sólo consideramos los cortes en $\not N$ (aunque podríamos habernos quedado sólo con el otro tipo de corte), debido a que como se observa en al Figura 5.5 no es necesario considerar el otro corte, ya que para otro de los desplazamientos cíclicos podremos conseguir un alineamiento equivalente. □

De acuerdo al Lema 5.5, el valor de $ATNLC_\gamma([x],[y])$ puede ser obtenido en tiempo $O(m^2n^2)$ realizando todos los desplazamientos cíclicos de las dos, resolviendo mn recurrencias del tipo que muestra (5.2).

Pero obviamente, necesitamos poder calcular el ATNLC para ponderaciones arbitrarias realizando el desplazamiento cíclico sobre sólo una de las cadenas. Para ello, deberemos redefinir de nuevo el ATNL, llamándolo en esta ocasión ATNL1, ya que sólo necesitaremos realizar el desplazamiento cíclico sobre una de las cadenas.

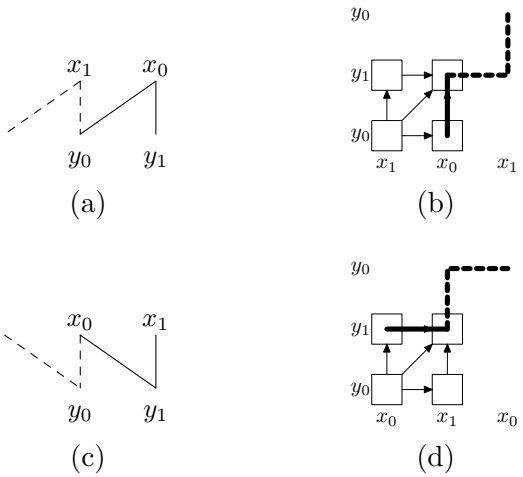


Figura 5.5: Ejemplo de alineamiento en el que se ven implicadas enes. Como se puede ver los alineamientos son equivalentes aunque sólo estamos considerando una de las enes en cada caso. En (a) y en (b), las líneas punteadas corresponden al caso en el que $i = m$ y $j = n$ en (5.2), en (c) y en (d) corresponderían también a este caso, suponiendo que hubiéramos considerado sólo la ene en lugar de la ene invertida. Aunque empezamos en (0, 0), la línea no aparece en este nodo para indicar que se empieza con un 0. (a) Alineamiento en el que se produce un corte en una ene invertida. (b) Grafo correspondiente al alineamiento de las cadenas que aparecen en (a). (c) Alineamiento en el que se produce un corte en una ene. (d) Grafo correspondiente al alineamiento de las cadenas que aparecen en (c).

Teorema 5.6 *Sea la ecuación recursiva para* $ATNL1_\gamma(x, y) = d(m, n)$*, donde*

$$d(i,j) = \begin{cases} 0, & \textit{si } i = j = 0; \\ d(i-1,j) + w(1,0) \cdot \delta(x_i, y_0), & \textit{si } i > 0 \ \textit{y} \ j = 0; \\ d(i,j-1) + w(0,1) \cdot \delta(x_0, y_j), & \textit{si } i = 0 \ \textit{y} \ j > 0; \\ \text{mín} \left\{ \begin{array}{l} d(i-1,j-1) + w(1,1) \cdot \delta(x_0, y_0), \\ d(i,j-1) + w(1,0) \cdot \delta(x_0, y_0) \end{array} \right\}, & \textit{si } i = m \ \textit{y} \ j = n; \\ \text{mín} \left\{ \begin{array}{l} d(i-1,j-1) + w(1,1) \cdot \delta(x_0, y_j), \\ d(i-1,j) + w(1,0) \cdot \delta(x_0, y_j), \\ d(i,j-1) + w(0,1) \cdot \delta(x_0, y_j) \end{array} \right\}, & \textit{si } i = m \ \textit{y} \ j \neq n; \\ \text{mín} \left\{ \begin{array}{l} d(i-1,j-1) + w(1,1) \cdot \delta(x_i, y_j), \\ d(i-1,j) + w(1,0) \cdot \delta(x_i, y_j), \\ d(i,j-1) + w(0,1) \cdot \delta(x_i, y_j) \end{array} \right\}, & \textit{en otro caso.} \end{cases} \tag{5.3}$$

La disimilitud ATNLC para ponderaciones arbitrarias entre una cadena cíclica $[x]$ *y una cadena cíclica* $[y]$ *puede calcularse con*

$$ATNLC_\gamma([x],[y]) = \min_{0 \leq k < m} ATNL1_\gamma(\rho^k(x), y).$$

Demostración: En (5.3) se unifican (5.1) y (5.2). Por un lado, tenemos el caso $i = m$ y $j \neq n$, en el que se realiza la concatenación del primer elemento de la cadena tal como ocurre en (5.1), es decir, añadimos una nueva columna al grafo como se muestra en la Figura 5.2c. Por el otro, está el caso $i = m$ y $j = n$, en el que se considera cómo se ha realizado la salida del grafo para entrar en el nodo $(0, 0)$. □

En la Figura 5.6 podemos ver el ejemplo de la Figura 5.4 con su solución utilizando el Lema 5.5 y el Teorema 5.6.

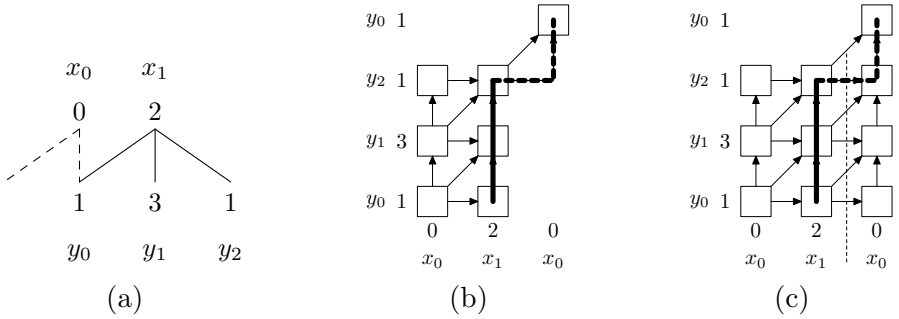


Figura 5.6: (a) Alineamiento óptimo entre las cadenas cíclicas $x = [02]$ e $y = [131]$, donde $\delta(x_i, y_j) = |x_i - y_j|$ y $\gamma = (1, 3, 1)$. (b) Grafo de alineamiento correspondiente a (a) que se obtiene utilizando (5.2). El grafo corresponde a la minimización del desplazamiento cíclico de las dos cadenas. Aunque empezamos en (0, 0), la línea no aparece en este nodo para indicar que se empieza con un valor o coste 0, como indica (5.2). (c) Grafo de alineamiento correspondiente a (a) que se obtiene utilizando (5.3). El grafo corresponde a la minimización del desplazamiento cíclico de la cadena x.

5.4. Algoritmo divide y vencerás

El valor de $ATNL(\rho^k(x), y)$ y $ATNL(\rho^k(x)x_k, y)$, para cada k, puede ser obtenido calculando los caminos más cortos en un grafo de alineamiento extendido (véase la Figura 5.8). Debido a que la propiedad de no cruce de los caminos de edición [Mae90] también se cumple con los caminos de alineamiento (Figura 5.9), el enfoque divide y vencerás puede ser aplicado al ATNLC.

Teorema 5.7 *Se puede calcular el ATNLC en tiempo $O(mn \log m)$ utilizando el algoritmo de la Figura 5.7.*

Demostración: Se debe tener en cuenta que el camino óptimo que empieza en $(k, 0)$ puede acabar en el nodo $(k + m - 1, n - 1)$ o en el nodo $(k + m, n - 1)$, es decir, tenemos dos caminos para cada inicio (véase la Figura 5.8). ¿Con cuál nos quedamos como límite para las siguientes recurrencias? En realidad es indiferente. Por la propiedad de no cruce de caminos los dos caminos que tenemos que calcular, en las recursividades, no van a poder cruzar ninguno de los caminos que tenemos como límite. El procedimiento divide y

vencerás sólo necesita de un camino como límite para izquierda o derecha con lo que por cada inicio sólo necesitaremos uno de los dos y no importa cuál. No obstante, para seguir un criterio unificado nos quedaremos con el de peso mínimo.

El tiempo de cómputo del algoritmo es $O(mn \log m)$: cada paso recursivo divide el espacio de búsqueda en dos mitades y todas las operaciones recursivas en el mismo nivel de recursividad requieren un tiempo total $O(mn)$. □

Figura 5.7: Algoritmo ATNLC.

Entrada: x, y: cadenas
Salida: $d^* : \mathbb{R}$
var P: vector $[0..m]$ de caminos de alineamiento
Inicio
  $d^* = \text{mín}(ATNL(\rho^0(x), y), ATNL(\rho^0(x)x_k, y)$
  Sea $P[0]$ el camino de alineamiento óptimo obtenido en el cálculo anterior
  Sea $P[m]$ igual a $P[0]$ desplazado m pasos a la derecha
  si $m > 1$ **entonces**
    $d^* = \texttt{mín}(d^*, \texttt{SiguientePaso}(x \cdot x, y, 0, m))$
  devolver d^*
Fin

función `SiguientePaso`(X: cadena, y: cadena, $l : \mathbb{N}$, $r : \mathbb{N}$):$\mathbb{R}$
Inicio
  $c = l + \lceil \frac{r-l}{2} \rceil$
  Sea $d = \text{mín}(ATNL(X_{c:c+m}, y), ATNL(X_{c:c+m+1}, y)$ conocidos $P[l]$ y $P[r]$
  si $l + 1 < c$ **entonces**
    $d = \texttt{mín}(d, \texttt{SiguientePaso}(X, y, l, c))$
  si $c + 1 < r$ **entonces**
    $d = \texttt{mín}(d, \texttt{SiguientePaso}(X, y, c, r))$
  devolver d
Fin

Para entender mejor la demostración podemos ver un caso extremo fijándonos en la Figura 5.8. Supongamos que estamos en la última recursividad y que sólo nos queda por calcular el camino que empieza en el nodo $(0, 1)$, con lo que tenemos los caminos que empiezan en $(0, 0)$ (con finales en $(3, 3)$ y $(3, 4)$) y $(0, 2)$ (con finales en $(3, 4)$ y $(3, 5)$) como límites. Se puede ver que los dos caminos que tenemos que calcular, en esta última recursividad, no van a poder cruzar ninguno de estos dos caminos límite a la izquierda y a la derecha.

En la Figura 5.10 podemos ver un ejemplo del procedimiento gráficamente. Tanto en el algoritmo como en el ejemplo nos quedamos como límites los caminos que terminan en $(k + m - 1, n - 1)$.

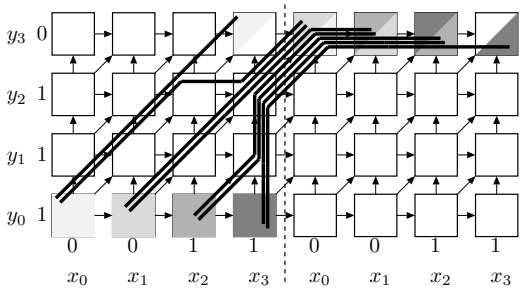


Figura 5.8: Grafo de alineamiento extendido para $x = 0011$ e $y = 1110$, donde $\delta(x_i, y_j) = |x_i - y_j|$. Los arcos que van al nodo (i, j) tienen un peso $\delta(x_i, y_j)$. El alineamiento óptimo para $[x]$ e $[y]$ es el camino de peso mínimo que empieza desde cualquier nodo coloreado en la fila de abajo y finaliza en un nodo que contenga el mismo color en la fila de arriba (se muestran todos los caminos candidatos con líneas gruesas).

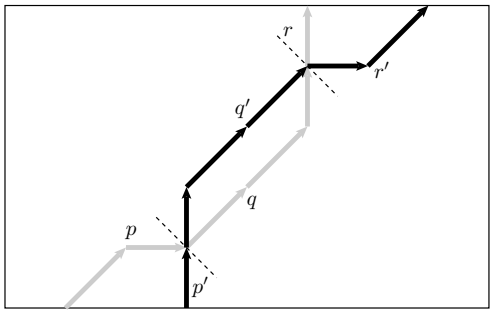


Figura 5.9: $P(j)$ es el camino óptimo de alineamiento para $\rho^j(x)$ e y, y $P(l)$ es el camino óptimo de alineamiento para $\rho^l(x)$ e y. El cruce de caminos puede ser evitado: si el peso de q es mayor que el peso de q', $P(j)$ puede ser mejorado escogiendo q' en vez de q.

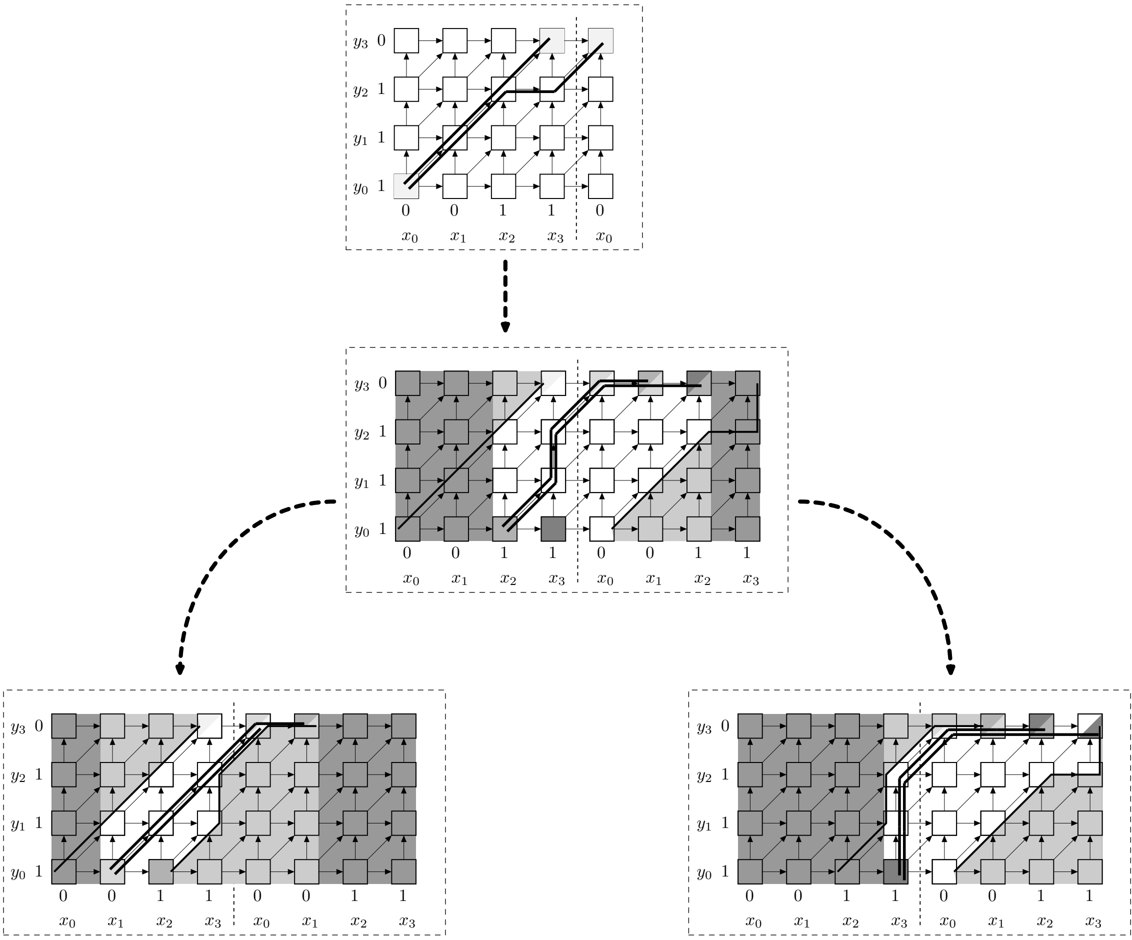


Figura 5.10: Procedimiento divide y vencerás para calcular el ATNLC entre las cadenas de la Figura 5.8. En primer lugar, se obtiene el alineamiento óptimo (camino) entre x e y y entre $\rho^0(x)x_0$ e y. El primer camino óptimo es utilizado como frontera izquierda y derecha en el grafo extendido: sólo se debe explorar la región blanca para el cálculo del camino óptimo entre $\rho^2(x)$ e y y entre $\rho^2(x)x_2$ e y. Esta idea se aplica recursivamente para calcular el resto de alineamientos óptimos, pero utilizando también el alineamiento óptimo entre $\rho^2(x)$ e y como una nueva frontera izquierda o derecha.

Hasta aquí con respecto al ATNLC con $\gamma = (1, 1, 1)$. Si queremos realizar el cálculo del ATNLC para ponderaciones arbitrarias, la extensión no es complicada. Sólo tendremos que trasladar al grafo extendido los casos específicos en los que se trata el corte de las enes en (5.3), el caso $i = j = 0$ y el caso $i = m$ y $j = n$, para cada uno de los desplazamientos cíclicos. En la Figura 5.11, se ilustra un ejemplo. El tratamiento de los caminos para el algoritmo divide y vencerás no se verá afectado.

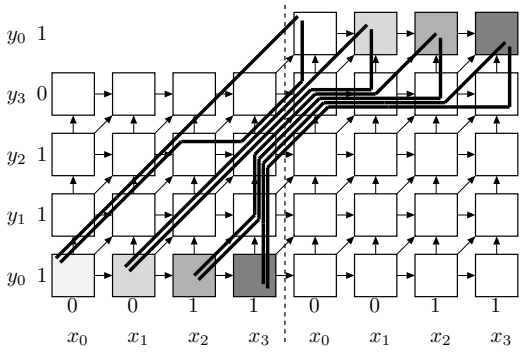


Figura 5.11: Grafo de alineamiento extendido para pesos arbitrarios. Con $x = 0011$ e $y = 1110$, donde $\delta(x_i, y_j) = |x_i - y_j|$ y $\gamma = (1, 1, 1)$. Los arcos que van al nodo (i, j) tienen un peso $\delta(x_i, y_j)$. El alineamiento óptimo para $[x]$ e $[y]$ es el camino de peso mínimo que empieza desde cualquier nodo coloreado en la fila de abajo y finaliza en un nodo que contenga el mismo color en la fila de arriba (se muestran todos los caminos candidatos con líneas gruesas).

5.5. Uso de restricciones globales

Como se ha explicado en la Sección 4.1.1 el uso de restricciones globales en el ATNL puede ser utilizado para acelerar el cálculo o para evitar caminos patológicos. Es por esto que parece también interesante extender su aplicación en el alineamiento de cadenas cíclicas.

Teorema 5.8 *La disimilitud ATNLC con la banda de Sakoe-Chiba entre una cadena cíclica $[x]$ y una cadena cíclica $[y]$ puede calcularse con:*

$$ATNLC_s([x], [y]) = \min_{0 \le k < m} \left(\min(ATNL_s(\rho^k(x), y), ATNL_s(\rho^k(x)x_k, y)) \right).$$

Demostración: Trivial a partir de la definición que aparece en la Sección 4.1.1 de la banda de Sakoe-Chiba y el Teorema 5.3. □

Teorema 5.9 *El algoritmo divide y vencerás (mencionado en la Sección 5.4) puede ser utilizado para obtener el ATNLC con la banda de Sakoe-Chiba en tiempo $O(mn \log m)$.*

Demostración: Para cada uno de los desplazamientos cíclicos (o cada recurrencia del algoritmo) haremos uso de una banda para la izquierda que se tendrá que combinar con el camino límite izquierdo, y una banda derecha que se tendrá que combinar con el camino límite derecho. Las combinaciones darán lugar a nuevos límites izquierdo y derecho. Obviamente para el límite izquierdo nos quedaremos con la posición que esté más a la derecha (del camino o de la banda izquierda) y para el límite derecho con la posición que esté más a la izquierda (del camino o de la banda derechas). Cuando tengamos los límites calculados podremos obtener el camino óptimo que empieza en $(k, 0)$ puede acabar o en el nodo $(k + m - 1, n - 1)$ o en el nodo $(k + m, n - 1)$. El tiempo de cómputo del algoritmo es finalmente $O(sn \log m)$. □

Aunque el coste computacional es menor que el cálculo del ATNLC sin banda, no estamos teniendo en cuenta que existe un cálculo adicional al combinar las bandas y los caminos límites en cada recurrencia. Experimentalmente se ha comprobado que el tiempo de computo sólo se ve reducido para valores de s pequeños.

Lo dicho anteriormente se puede aplicar también a una ponderación γ arbitraria y/o a la banda de Itakura.

5.6. ATNLC Normalizado

Por lo comentado en la Sección 4.1.3 parece claro que la normalización también puede ser útil para el alineamiento de cadenas cíclicas.

Utilizando el ATNL1 normalizado (ATNL1N), podemos definir el ATNLC normalizado (ATNLCN), de $[x]$ e $[y]$ con una ponderación γ, como

$$ATNLCN_\gamma([x],[y]) = \min_{0\le k<m} ATNL1N_\gamma(\rho^k(x), y) = \min_{x\in[x]} \min_{p\in\mathcal{P}} \frac{W(p)}{L(p)}.$$

Basándonos en la Sección 3.2, los problemas pasan a ser:

Problema N.

$$d^* = \min_{x\in[x]} \min_{p\in\mathcal{P}} \frac{W(p)}{L(p)}, \quad W, L : \mathcal{P}; \quad L(p) > 0, \ \forall p \in \mathcal{P}.$$

El Problema N se puede resolver por medio del problema $N(\lambda)$:

Problema N(λ).

$$d^*(\lambda) = \min_{x\in[x]} \min_{p\in\mathcal{P}} (W(p) - \lambda L(p)), \quad W, L : \mathcal{P}; \quad L(p) > 0, \ \forall p \in \mathcal{P}.$$

Utilizando el algoritmo divide y vencerás el coste computacional es de $O(tmn \log(m))$, siendo t el número de iteraciones. Recordemos que con la distancia de edición este número es muy bajo (entre 2 y 4 iteraciones). En este caso, ocurre lo mismo.

5.7. Discusión

En este capítulo hemos formalizado el ATNLC. Hemos demostrado con contraejemplos que una extensión inmediata del algoritmo de Maes al alineamiento no es correcta, contrariamente a lo que suponen diferentes autores en la literatura. También hemos presentado ecuaciones recursivas y algoritmos que combinan la estrategia divide y vencerás con la programación dinámica para efectuar el cálculo del alineamiento cíclico óptimo. Por otro lado, también se han combinado estas técnicas con las bandas de Sakoe y la normalización.

El algoritmo divide y vencerás presenta un ahorro temporal importante, aunque en ciertas situaciones esto no es suficiente. En en el Capítulo 6 veremos que es posible acelerar todavía más las comparaciones en tareas de reconocimiento.

En el Capítulo 8, se realizará un estudio experimental del ATNLC con diversos descriptores de contorno, entre otros experimentos.

Capítulo 6

Aceleración del reconocimiento

Porco Rosso: «*A pig's gotta fly.*»
Hayao Miyazaki, *Pòrco Rósso* (1998).

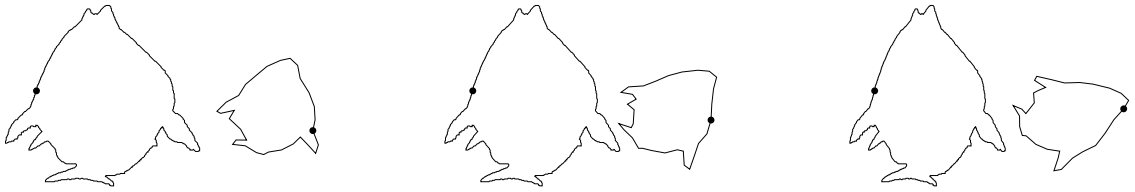
Existen dos enfoques para acelerar el reconocimiento de cadenas cíclicas. El primero consiste en acelerar el cálculo de la propia distancia (en nuestro caso el ATNLC, como se ha hecho en el anterior capítulo). El segundo consiste en la utilización de algún método de organización (o indexado) de nuestro conjunto de prototipos que nos permita no tener que realizar la exploración de todo este conjunto [CNBYM01, Het].

En este capítulo trataremos de dar soluciones con estos dos enfoques.

6.1. Heurístico de alineamiento con patrón de referencia: APR

En la literatura se suelen utilizar las propiedades del espacio frecuencial (véase la Sección 2.2.7) para encontrar una rotación de referencia y con esta un punto inicial (Sección 2.4.1). La invarianza a la rotación puede ser obtenida restando $(\theta_{-1} + \theta_1)/2$ (la orientación de la elipse principal) a todos los θ_k. La invarianza al punto de inicio se puede conseguir sumando $k(\theta_{-1} - \theta_1)/2$ a todos los θ_k. De esta manera, sólo habría que realizar un ATNL y no un ATNLC para comparar las cadenas. Sin embargo, usar la orientación de la elipse principal presenta una ambigüedad π radianes [FS02] (aunque se puede usar algún heurístico para tratar de resolverla [HK91] o usar las dos cadenas, la «normal» y su rotación de 180 grados, para representar el contorno, lo que en reconocimiento supone efectuar dos comparaciones en lugar de sólo una) y presenta problemas con formas que tienen una elipse principal cercana a la circularidad. En la Figura 6.1 puede verse un ejemplo, donde ligeras modificaciones hacen que la rotación y el punto inicial cambien considerablemente. Además, si las figuras de una categoría tienen diferencias importantes (por ejemplo, la figura de una persona con un brazo abierto y la figura de otra persona con los brazos tocando el cuerpo) podrán hacer que la orientación sea diferente. Estas consideraciones son obviadas, por ejemplo, en trabajos como [BCP05].

En el caso de que tengamos una tarea de reconocimiento en la que los contornos estén etiquetados por su pertenencia a una categoría, como por ejemplo, en tareas de



(a) (b) (c)

Figura 6.1: (a) Forma original y su normalización. (b) La misma forma con una ligera compresión en el eje x y su normalización. (c) La misma forma un poco más comprimida y su normalización.

clasificación, podemos aplicar un heurístico que se comporta mejor, según se verá en el capítulo de experimentos.

Se necesita, en primer lugar, hacer un preproceso. Partiendo de un conjunto de prototipos etiquetados, nuestro objetivo consiste en escoger un buen punto inicial para estos. Para ello, seleccionaremos un representante en cada categoría (que puede ser el centroide) y un punto cualquiera inicial para cada uno. Con el representante de cada categoría y su punto inicial, calcularemos el desplazamiento cíclico de los demás miembros de la categoría que necesitamos para obtener el ATNLC entre estos. Con este fin podemos utilizar el algoritmo que aparece en la Figura 6.3, una versión modificada del que aparece en la Figura 5.7, para devolver además el desplazamiento. El procedimiento de preproceso se muestra en la Figura 6.2.

Figura 6.2: Algoritmo de preproceso.

Entrada: P: prototipos
Salida: P: prototipos con nuevo desplazamiento cíclico
Inicio
  para $p \in P$ **hacer**
    d, $desp = ATNLCD(p, \texttt{Representante}(p))$
    $p = \rho^{\text{desp}}(p)$
  devolver P
Fin

Una vez tenemos los prototipos preparados, podemos ya emplearlos en una tarea de clasificación[1]. Para ello, utilizaremos de nuevo el representante de cada categoría para obtener un buen punto inicial. Cada vez que se quiera clasificar una muestra, en el caso de una búsqueda exhaustiva, mediremos las distancias entre esta muestra y los prototipos en orden de categorías, es decir, primero los prototipos de una categoría, luego los de la siguiente y así de manera sucesiva. De este modo, sólo se tendrá que hacer un ATNLCD con el representante de cada categoría, consiguiendo hacer el máximo número de ATNL no cíclicos.

[1]Aunque, en principio, se podría extender a cualquier tarea de reconocimiento en las que tengamos categorías definidas.

Figura 6.3: Algoritmo ATNLCD.

Entrada: x, y: cadenas
Salida: d^*: $\mathbb{R}$, *desp*: $\mathbb{N}$ // Distancia y desplazamiento cíclico
var P: vector $[0..m]$ de caminos de alineamiento
Inicio
  $d^* = \text{mín}(ATNL(\rho^0(x), y), ATNL(\rho^0(x)x_0, y)$
  Sea $P[0]$ el camino de alineamiento óptimo obtenido en el cálculo anterior
  Sea $P[m]$ igual a $P[0]$ desplazado m pasos a la derecha
  si $m > 1$ **entonces**
    $d =$ `SiguientePaso(`$x \cdot x$, y, 0, m, *desp*`)`
    si $d^* < d$ **entonces**
      desp $= 0$
    si no
      $d^* = d$
  devolver d^*, *desp*
Fin

función `SiguientePaso(`X*:* cadena, y*:* cadena, $l : \mathbb{N}$, $r : \mathbb{N}$, *rdesp:* $\mathbb{N}$`)` : $\mathbb{R}$
Inicio
  $c = l + \lceil \frac{r-l}{2} \rceil$
  desp $= c$
  Sea $d = \text{mín}(ATNL(X_{c:c+m}, y), ATNL(X_{c:c+m+1}, y)$ conocidos $P[l]$ y $P[r]$
  si $l + 1 < c$ **entonces**
    $dl =$ `SiguientePaso(`X, y, l, c, *desp*`)`
    si $dl < d$ **entonces**
      $d = dl$
      rdesp $=$ *desp*
  si $c + 1 < r$ **entonces**
    $d =$ `mín(`d, `SiguientePaso(`X, y, c, r, *desp*`))`
    si $dr < d$ **entonces**
      $d = dr$
      rdesp $=$ *desp*
  devolver d
Fin

En el caso de otro tipo de búsquedas, de manera genérica, podemos medir el ATNLCD cuando se necesite y guardarlo para otros prototipos de la misma categoría. Es decir, antes de medir la distancia ATNL entre la muestra y un prototipo, realizaremos sobre la muestra el desplazamiento cíclico (de la categoría del prototipo) si lo tenemos almacenado y si no, lo calcularemos, almacenándolo por si se necesita posteriormente para otro prototipo de la misma categoría.

Este heurístico reduce el coste computacional y da pie a utilizar las técnicas de aceleración del ATNL, comentadas en la Sección 4.2. Ahora bien, como veremos (en el capítulo de experimentos) las tasas de reconocimiento son ligeramente peores respecto a la utilización del ATNLC para calcular todas las distancias.

6.2. Aceleración del ATNLC en la obtención de los k-vecinos

El algoritmo divide y vencerás propuesto en la Sección 5.4 reduce mucho el tiempo de cálculo del ATNLC respecto a la fuerza bruta. Ahora bien, si nuestro propósito es el reconocimiento de formas será posible reducir todavía más este tiempo. Tanto en la clasificación como en la recuperación de formas se puede realizar una búsqueda exhaustiva de los k-vecinos más próximos a una muestra dada. Sólo que en el caso de la clasificación la k suele ser pequeña y en la recuperación la k es grande. Si esto es así, si antes o durante el cálculo de una distancia podemos saber que esta no va a mejorar al último de los k-vecinos más próximos (calculados hasta ese momento), podemos no hacer o parar este cálculo. Al valor de distancia del último de los k-vecinos más próximos le llamaremos cota externa.

En la Sección 3.3.3 se habló de una técnica de ramificación y poda para hacer el cálculo de la DEC. En ella se hace uso de cotas inferiores basadas en el desplazamiento cíclico de la cadena. Para la determinación de estas cotas se utiliza lo siguiente. La diferencia entre la distancia de edición entre dos cadenas y la distancia de edición entre una de estas cadenas y un desplazamiento cíclico queda acotada por una inserción y un borrado [MG01]:

$$|DE(\rho^{k+1}(x), y) - DE(\rho^k(x), y)| \leq \gamma_I + \gamma_D.$$

Sin embargo, esto no se puede aplicar al ATNLC. Cuando realizamos un desplazamiento cíclico en una de las cadenas esta diferencia no se puede acotar de una manera tan sencilla, ya que el desplazamiento puede trastocar completamente el alineamiento y por lo tanto el valor de la disimilitud. Quizás se vea más claro con el siguiente ejemplo. Supongamos dos cadenas $x = 1110$ e $y = 0001$, donde $ATNL(x, y) = ATNL(1110, 0001) = 3$, siendo $\delta(x_i, y_i) = |x_i - y_i|$. Si realizamos un desplazamiento cíclico de la cadena x pasará lo siguiente, $ATNL(\rho^1(x), y) = ATNL(0111, 0001) = 0$. En el ATNL no tenemos ni borrados ni inserciones, sólo alineamientos y no parece posible acotarlos con una operación que por lo menos sea lineal.

Tal como se explicó en la Sección 3.3.3, la técnica de ramificación y poda desestima el cálculo entre una limitación entre dos caminos óptimos (que no pueden cruzarse por otro camino óptimo) cuando la cota inferior es mayor que la cota externa, lo cual para la DEC resulta en un método extremadamente rápido para el reconocimiento. No obstante, como hemos visto, esta técnica no se puede utilizar con el ATNLC por lo cual deberemos buscar otras vías.

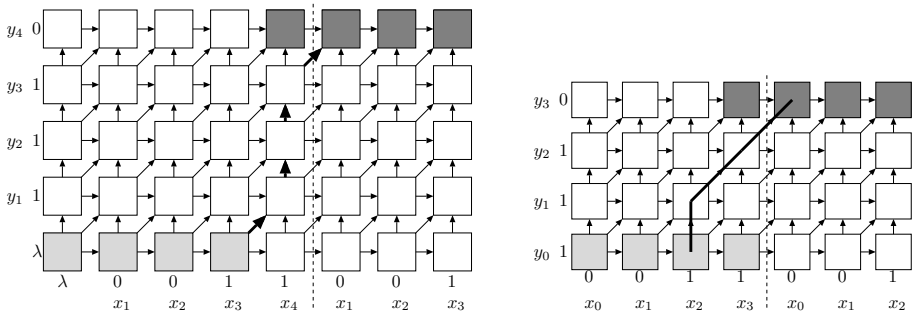


Figura 6.4: (a) Cálculo del BB en el grafo de edición extendido para $x = 0011$ e $y = 1110$. Los aciertos tienen un peso 0 y cualquier otra operación tiene peso 1. El algoritmo BB obtiene 1 como un valor aproximado de $DEC([x],[y])$ y su camino se muestra con flechas gruesas. El camino debe de empezar en cualquier nodo gris claro (fila inferior) y terminar en un nodo de color gris oscuro (fila superior). (b) Un cálculo equivalente del algoritmo BB para la DEC para estimar una aproximación del ATNLC en el grafo de alineamiento extendido para x e y, donde $\delta(x_i, y_j) = |x_i - y_j|$. El alineamiento resultante se muestra con una línea gruesa.

6.2.1. Pseudo-alineamiento en el grafo de alineamiento extendido

Como se vio en la Sección 3.3.4 el algoritmo de Bunke y Bühler [BB93] aproxima el valor de la DEC con un pseudo-alineamiento. Este pseudo-alineamiento se calcula, sobre el grafo extendido, inicializando todos los nodos de partida con el valor 0 y calculando el valor mínimo de cualquier alineamiento óptimo que llegue a un nodo final (véase la Figura 3.9a).

Para poder aplicar esta aproximación al ATNLC utilizaremos un grafo de alineamiento extendido (véase Figura 6.4b). Partiremos de cualquier nodo $(i, 0)$ para $0 \leq i < m - 1$, y llegaremos a cualquiera de los nodos $(i, n-1)$, para $m-1 \leq i < 2m-1$. En principio, los nodos de partida deberán inicializarse, en vez de con 0, con $\delta(x_i, y_o)$.

No obstante, como el siguiente contraejemplo muestra, no lo podemos hacer así: sea $\delta(x_i, y_j) = |x_i - y_j|$; el valor de $ATNLC([x],[y]) = ATNLC([101],[010])$ es 0, ya que $DTW(110, 100) = 0$. Si calculamos el BB para todo posible desplazamiento cíclico de x, $BB(101, 010) = BB(011, 010) = 0$, pero $BB(110, 010) = 1$. Esto contradice el hecho de que $BB(x, y) \leq ATNLC([x],[y]])$ (tal como se comenta en la Sección 3.3.4), ya que $CDTW([010],[101]) < BB(110, 010)$ (véase la Figura 6.5a).

El pseudo-alineamiento BB del ATNLC (a partir de ahora BBATNL) entre dos cadenas cíclicas $[x]$ e $[y]$ puede ser calculado buscando el camino óptimo que parte de cualquier nodo $(i, 0)$ para $0 \leq i < m-1$, y llega a cualquier nodo $(i, n-1)$, para $m-1 \leq i \leq 2m-1$. De acuerdo al Teorema 5.3, si añadimos una columna en el grafo extendido de alineamiento y utilizamos estos nodos de partida y de llegada, podemos encontrar todos los caminos correspondientes a $ATNL(\rho^k(x), y)$ y $ATNL(\rho^k(x)x_k, y)$ for $0 \leq k < m$. Por lo tanto, el camino correspondiente al pseudo-alineamiento está también incluido. En la Figura 6.6 podemos ver, gráficamente, lo que hemos comentado. Si nos fijamos en la Figura 6.5b veremos que el contraejemplo deja de serlo.

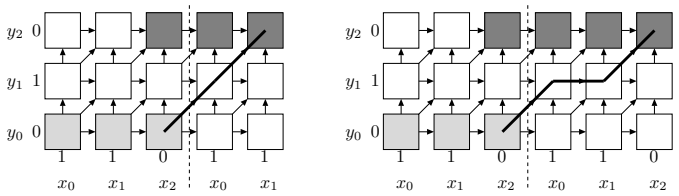


Figura 6.5: Contraejemplo: (a) Cálculo del BB (para la DEC) para estimar el ATNLC para $x = 001$ and $y = 010$, donde $\delta(x_i, y_j) = |x_i - y_j|$. El valor obtenido es 1, mayor que $ATNLC([x], [y]) = 0$. (b) El cálculo correcto de BB (BBATNL) para la estimación del ATNLC. El valor obtenido es 0.

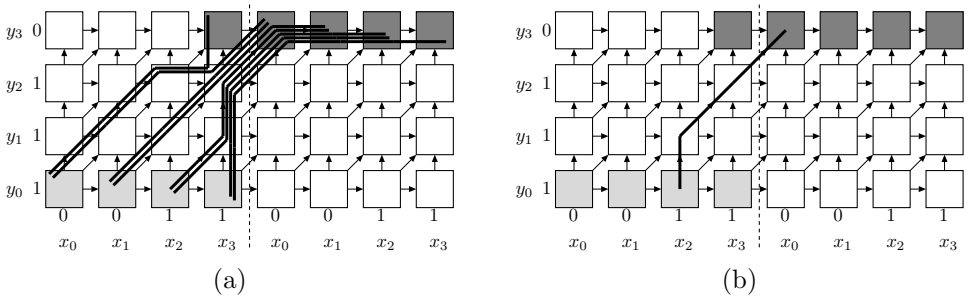


Figura 6.6: (a) Podemos ver aquí todos los caminos correspondientes a $ATNL(\rho^k(x), y)$ y $ATNL(\rho^k(x)x_k, y)$ para $0 \leq k < m$, que parten desde cualquier nodo $(k, 0)$ para $0 \leq k < m$ (nodos gris claro), y llegan a cualquier nodo $(k', n-1)$, para $m-1 \leq k' < 2m$ (nodos gris oscuro). (b) Camino obtenido por el BBATNL que aproxima la disimilitud ATNLC en el grafo de alineamiento extendido para x e y, donde $\delta(x_i, y_j) = |x_i - y_j|$.

De este modo.

Teorema 6.1 *La estimación que produce el BBATNL es una cota inferior del ATNLC:*

$$BBATNL(x, y) \leq ATNLC([x], [y]).$$

Demostración: Trivial a partir de lo anterior. □

Todo lo explicado en esta sección es para la ponderación $\gamma = (1, 1, 1)$. Si queremos aplicarlo a ponderaciones arbitrarias habrá que añadir los casos pertinentes, tal como se explica en la Sección 5.4.

6.2.2. Cota inferior basada en BBATNL

Así, cuando utilizamos el ATNLC en reconocimiento, podemos reducir el coste computacional con el BBATNL como una función de cota inferior. No necesitaremos ejecutar el ATNLC cuando el BBATNL nos dé un valor mayor que la cota externa.

El BBATNL necesita un tiempo $O(mn)$ para ser calculado. Sin embargo, es posible todavía acelerar más el proceso. Estamos interesados en el calculo de BBATNL ya que

su valor nos puede permitir (haciendo uso de la cota externa) no tener que calcular el ATNLC. Por esto, podremos parar la ejecución de $BBATNL(x, y)$ cuando sepamos que su valor va a ser mayor que la cota externa. A esto se le llama abandono prematuro (*early abandon* [AFS93]). La forma de obtener esto es parar en la fila donde todos los valores son mayores que esta cota externa, ya que si esto ocurre todos los caminos que lleguen a los nodos finales van a ser mayores. El algoritmo del BBATNL que hace uso de la cota externa se muestra en la Figura 6.7.

Figura 6.7: Algoritmo BBATNL con cota externa.

Entrada: x, y: cadenas, *cotaexterna*: $\mathbb{R}$
Salida: *distancia*: $\mathbb{R}$
var M: matriz $[0\,..\,2m-1][0\,..\,n-1]$ de $\mathbb{R}$
Inicio
  para i **en** $0\,..\,m-1$ **hacer**
    $M[i][0] = \delta(x[i], y[0])$
  para i **en** $m\,..\,2m-1$ **hacer**
    $M[i][0] = M[i-1][0] + \delta(x[i], y[0])$
  para j **en** $1\,..\,n-1$ **hacer**
    $M[0][j] = M[0][j-1] + \delta(x[0], y[j])$
  para i **en** $1\,..\,2m-1$ **hacer**
    para j **en** $1\,..\,n-1$ **hacer**
      $M[i][j] = \texttt{mín}(M[i-1][j-1], M[i-1][j], M[i][j-1]) + \delta(x[i], y[j])$
    si todo elemento de la fila i es $\geq$ *cotaexterna* **entonces**
      distancia $= \infty$
      devolver *distancia*
  distancia $=$ `mín` $(M[i][n-1]$ **para** i *en* $m-1\,..\,2m-1)$
  devolver *distancia*
Fin

Este algoritmo puede ser interesante también si el coste de las distancias locales (el coste de δ) es muy grande, como ocurre con los descriptores mencionados en la Sección 2.2.11. En este caso, el cálculo de las distancias locales tiene un peso muy grande en el coste total y es muy importante eliminar cuantas nos sea posible. Para ello, se irán calculando de manera incremental en el BBATNL, almacenándolas en una matriz $m \times n$. Así, si en el BBATNL paramos en la fila k, nos habremos ahorrado el cálculo de $(n-k) \cdot m$ distancias locales, y si llegamos al final, la matriz estará completamente calculada y se la podremos pasar al ATNLC para que no la tenga que volver a calcular. De aquí en adelante a esta composición de los dos algoritmos la llamaremos BBATNLC.

6.3. Aceleración de la búsqueda de vecinos con AESA

En determinadas situaciones, realizar una búsqueda sobre todo el conjunto de prototipos no resulta práctico. En estos casos, lo que se suele hacer es utilizar métodos de acceso basados en espacios métricos. Estos métodos organizan (o indexan) la base de datos de una forma en la que las consultas de similitud pueden realizarse eficientemente sin la necesidad de procesar toda la base de datos. De manera genérica, el principio por el cual se rigen es la desigualdad triangular (propiedad que satisface cualquier métrica) y con ella se organizan los prototipos en clases diferentes. Cuando se realiza una consulta sólo se busca en las clases candidatas, es decir, aquellas que solapan con la consulta (por así decirlo), y así la búsqueda es más eficiente. Existen muchos métodos de este estilo: *M-tree*, *R-tree*, *vp-tree*, ... [CNBYM01], aunque de entre todos ellos, destacan los algoritmos basados en AESA (*Approximating and Eliminating Search Algorithm*) [Vid86, Vid94, MOV94], cuando el cálculo de una distancia entre prototipos es muy costosa (véase la Sección 4.2). Este es precisamente nuestro caso, por dos motivos, el cálculo del ATNLC y el cálculo de las distancias locales que en ocasiones, como hemos visto, puede tener un peso importante en el coste total.

Los métodos basados en AESA, como hemos dicho anteriormente, trabajan en espacios métricos. Pero esto desgraciadamente no ocurre con el ATNL. Tal como nos muestran en [VCR85, Lem09] con sendos contraejemplos, el ATNL no cumple la desigualdad triangular. De aquí podemos derivar que el ATNLC tampoco. Más adelante (en la Sección 6.3.3) explicaremos cómo, en parte, se puede paliar este problema.

Seguramente este algoritmo es el que menos distancias calcula para obtener el vecino más próximo. Pero tiene un gran inconveniente, su complejidad espacial es cuadrática debido a la matriz de distancias que se calcula en el preproceso. Para solucionar este problema surgió LAESA (*Linear AESA*) [MOV94]. En el preproceso se elige, a partir del conjunto de prototipos, un conjunto B de prototipos base. Se calcula la distancia de cada prototipo base a todos los demás del conjunto de prototipos y se almacena en una matriz que ya no es cuadrada, con lo que la complejidad espacial del LAESA pasa a ser lineal con respecto al conjunto de prototipos y la cardinalidad de B.

En el proceso de clasificación, se seleccionan como candidatos a vecino más próximo los prototipos base, y se calcula su distancia a la muestra, utilizando dicha distancia y las distancias calculadas en el preproceso, para obtener una cota inferior de la distancia de cada prototipo a la muestra utilizando la desigualdad triangular (como en el AESA). Cuando se han eliminado todos los prototipos base[2], sigue el mismo procedimiento que con el AESA, pero sin actualizar las cotas.

El número de distancias que calcula el LAESA es siempre mayor que el que calcula el AESA (para que fueran iguales debería de suceder que $B = P$, es decir, el conjunto de prototipos base tendría que ser todo el conjunto de prototipos). Pero empíricamente [Mic96] se ha demostrado que la cantidad de distancias en ambos algoritmos crece muy lentamente con respecto al tamaño del conjunto de entrenamiento, es decir, no depende de este

[2]En este trabajo se ha utilizado la aproximación EC1 [MOV94], tal como se hace en trabajos posteriores de los mismos autores [JM03, MSMO03]. En esta aproximación se calculan todas las distancias de la muestra a los prototipos base inicialmente sin podarlos.

tamaño. Ahora bien, un factor importante del que si dependen las distancias calculadas es el número de prototipos de B y la selección de estos. En [Mic96] se hizo un estudio a este respecto y se llegó a la conclusión de que el número de prototipos base depende de la dimensionalidad de la base de datos, y se deben seleccionar de forma que estén separados lo máximo posible.

6.3.1. Una versión de LAESA con uso de cota externa: LAESAEA

Como se ha comentado, en LAESA sólo se realiza la actualización de las cotas inferiores con las distancias calculadas para todos los prototipos que pertenecen a B. Cuando estamos calculando la distancia a un prototipo que no pertenece a B, no es necesario calcular las distancias porque no necesitamos actualizar las cotas. De este modo, podemos parar cuando sepamos que esta distancia no va a ser menor que la del vecino más cercano. En el caso de que el prototipo pertenezca a B, calcularemos la distancia tal cual, es decir, ATNLC, y si no pertenece a B podremos hacer uso de BBATNLC. Los prototipos que no se poden con la cota inferior de la desigualdad triangular, se pueden podar ahora con una cota inferior más ajustada, el BBATNLC.

A este nuevo algoritmo lo hemos llamado LAESAEA (*LAESA with Early Abandon*). Obviamente, esto no es aplicable a AESA ya que $B = P$, y por tanto, tenemos que calcular todas las distancias para actualizar las cotas.

En la Figura 6.8 se muestra el algoritmo LAESAEA para el caso del ATNLC. Las modificaciones realizadas para pasar de LAESA a LAESAEA están comentadas. Para transformar el algoritmo a LAESA, sólo tenemos que eliminar la condición (que contiene comentarios) en la que se comprueba si $s \in B$, y calcular siempre la distancia completa. Y por último para pasar de LAESA a AESA, como se ha dicho, igualar B a P, $B = P$.

Aunque hasta ahora se ha hablado sólo del vecino más próximo, AESA, LAESA y LAESAEA pueden extenderse a los k-vecinos más próximos (para una mejor clasificación -según el caso- o la recuperación de formas). Para ello, sólo hay que mantener una lista ordenada de los k-mejores y utilizar como cota externa la distancia al último de estos, de la misma forma que se ha comentado en la Sección 6.2 [Vid86].

6.3.2. Dimensionalidad intrínseca y su impacto en el coste de la búsqueda de vecinos

Los métodos de indexado basados en métricas no necesariamente funcionan con todas las bases de datos y todas las métricas. Su eficiencia se ve afectada por la distribución de distancias en la base de datos. A partir de esta distribución podemos obtener la dimensionalidad intrínseca. Según [CNBYM01], dada una base de datos D y una métrica m, la dimensionalidad intrínseca ϱ viene dada por $\varrho(D,m) = \frac{\mu^2}{2\sigma^2}$, donde μ y σ^2 son la media y la varianza de la distribución de distancias. En [CNBYM01] se muestra de manera analítica y experimental que todos los algoritmos basados en métricas se degradan de manera sistemática a medida que incrementa ϱ, es decir, se va acercando al coste temporal de una búsqueda exhaustiva [WSB98].

Si miramos la fórmula, podemos ver que la dimensionalidad intrínseca aumenta por las dos siguientes razones: decrece la varianza y/o aumenta la media de la distribución de

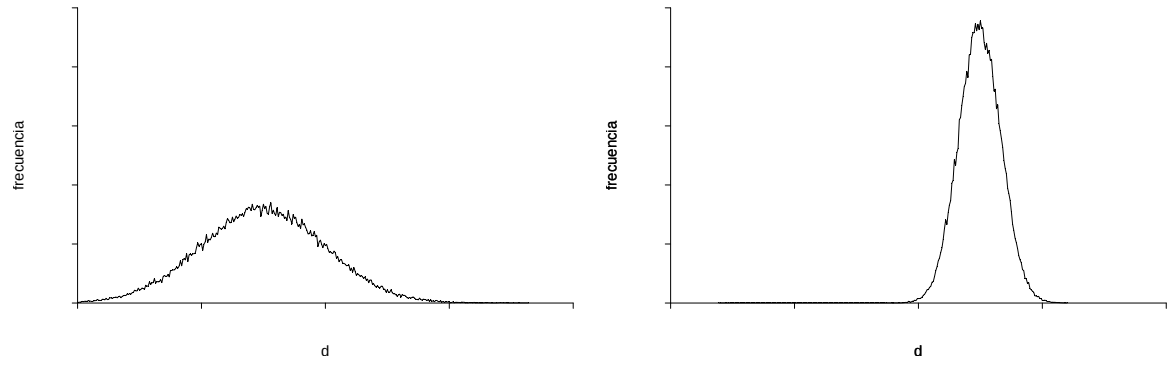
Figura 6.8: Algoritmo LAESAEA para ATNLC.

Entrada: P: prototipos, x: muestra a clasificar
Salida: $nn \in P$: vecino más cercano
var $B \subset P$: prototipos base, $D_B \in \mathbb{R}^{|B|\times|P|}$: distancias a prot. base
Inicio
  para $p \in P$ **hacer**
    $G[p] = 0$
  nn = desconocido; $d_{nn} = \infty$
  s = elemento cualquiera de B
  mientras $|P| > 0$ **hacer**
    si $s \in B$ **entonces**
      $d_s = ATNLC(x, s)$ // calculamos la distancia tal cual
    si no
      $d_s = BBATNLC(x, s, d_{nn})$ // paramos el cálculo si no mejoramos a d_{nn}
    $P = P - s$
    si $d_s < d_{nn}$ **entonces**
      $nn = s$; $d_{nn} = d_s$
    $siguiente_B$ = desconocido; $gmin_B = \infty$
    $siguiente$ = desconocido; $gmin = \infty$
    para $p \in P$ **hacer**
      si $s \in B$ **entonces**
        $G[p] = \texttt{máx}(G[p], |D_B[s, p] - d_s|)$
      si $G[p] > d_{nn}$ **entonces**
        $P = P - p$
      si no
        si $p \in B$ **entonces**
          si $G[p] < gmin_B$ **entonces**
            $gmin_B = G[p]$; $siguiente_B = p$
        si no
          si $G[p] < gmin$ **entonces**
            $gmin = G[p]$; $siguiente = p$
    si $siguiente_B \neq$ desconocido **entonces**
      $s = siguiente_B$
    si no
      $s = siguiente$
Fin

distancias. En la Figura 6.9 pueden verse dos distribuciones de distancias mostrando una baja ($\varrho = 4$) y una alta ($\varrho = 68$) dimensionalidad intrínseca. Dos casos extremos, en los que varían tanto la varianza como la media.

Si se reduce la varianza querrá decir que mayor cantidad de distancias tienen similar valor, con lo cual vamos a tener cada vez menos información para realizar las podas (en el caso de AESA las cotas van a ser peores). Por otro lado, si aumenta la media de la distribución para obtener los vecinos más próximos deberemos explorar más prototipos (concretamente en AESA podemos tardar más en encontrar un prototipo que realice una buena poda).

Ahora bien, de qué depende la distribución de distancias en nuestro problema, es decir, qué provoca que ϱ incremente. Podemos considerar dos factores principales. Uno de ellos es la cadena cíclica o descriptor de contorno (en nuestro caso), afectando especialmente el número de puntos y el número de dimensiones que estemos utilizando para cada punto. Por ejemplo, en el descriptor BAS (véase la Sección 2.2.11.2) se utilizan 4 dimensiones por punto y en los *shape contexts* 60 dimensiones por punto (véase la Sección 2.2.11.1). El segundo motivo es la distancia que utilicemos para comparar estas cadenas. Aunque si fijamos como distancia el ATNLC cobra importancia la distancia local, δ, que con el descriptor BAS es la distancia euclídea y con los *shape contexts* es χ^2 [BMP02].



(a) (b)

Figura 6.9: Ejemplo sintético de dos distribuciones de distancias. (a) Con una dimensionalidad intrínsa baja, $\varrho = 4$. (b) Con una dimensionalidad intrínseca alta, $\varrho = 68$.

6.3.3. Sobre la desigualdad triangular y la dimensionalidad intrínseca

En la Sección 2.3.1 se comentaron las propiedades que debía de cumplir una distancia para ser una métrica. Si en el ATNL se utilizan producciones simétricas (véase la Sección 4.1.2), la única propiedad problemática es la desigualdad triangular:

$$d(x, z) \leq d(x, y) + d(y, z),$$

ya que se pueden encontrar contraejemplos donde el ATNL no la cumple [VCR85, Lem09]

y por tanto no es una métrica. Como hemos dicho la corrección de algoritmos como AESA dependen de que se observe esta propiedad.

En [VCR85, CVR87, VRCB88, VCBL88] se realizó un estudio con una tarea de reconocimiento de voz con palabras aisladas [RJ93] utilizando ATNL. Se pretendía ver cómo afectaba el incumplimiento de la desigualdad triangular en muestras del mundo real. Estas muestras eran tramas de voz que se representaban con cadenas donde cada componente tenía 8 dimensiones. En [CVR87], en 15 millones de tripletas no se encontró ningún caso donde se violase la desigualdad triangular. En [Lem09] se realizaron también experimentos similares con series temporales (cadenas con elementos de una dimesión) sintéticas de tres tipos: series temporales de *white-noise*, *random-walk* y *cylinder-bell-funnel* [Sai00]. El tipo más problemático fue el *random-walk* (con los otros tipos sólo se encontró una tripleta problemática) donde se llega hasta a un 20 % de tripletas que no cumplen la desigualdad triangular.

Para ver cuántas tripletas x, y, z no cumplen la desigualdad triangular se puede utilizar la siguiente fórmula:

$$H = d(x, y) + d(y, z) - d(x, z). \tag{6.1}$$

Todas las tripletas que tengan una H menor que cero no cumplen la desigualdad triangular. En [VCR85] se muestran distribuciones (o histogramas) de las frecuencias de aparición de tripletas para cada H. Estas distribuciones parecen tener forma de gaussianas donde para $H = 0$ la frecuencia es ya muy baja.

En (6.1) se puede ver que la distribución de H guarda relación con la distribución de distancias (comentada en la Sección 6.3.2). Es decir, H es una composición de tres variables aleatorias (dependientes entre sí) con la misma distribución. Cuanto más grande sea la media en la distribución de distancias, los valores de H de la mayoría de las tripletas serán más altos, y por tanto, habrá más valores positivos, ya que estaremos sumando dos pares de distancias de la misma distribución y restando otra, también de la misma. Con la varianza ocurrirá más de lo mismo pero cuanto más baja sea, debido a que las distancias serán más parecidas, y por tanto, habrán más valores de H mayores o iguales a cero.

Así, podemos conjeturar que, cuanto más alta sea la dimensionalidad intrínseca, ϱ, muy probablemente nos encontraremos con menos tripletas, x, y, z, que incumplan la desigualdad triangular.

En la práctica, y en el caso del ATNLC, esta afirmación nos dice, por ejemplo, que va a ser más sencillo encontrar tripletas que no cumplen la desigualdad triangular en conjuntos de cadenas cuyos elementos tengan una dimensión, como la curvatura (véase la Sección 2.2.2.4), que en conjuntos con cadenas de cuatro dimensiones, como el descriptor BAS (véase la Sección 2.2.11.2). Aunque también intervendrán otros factores, como los propios datos, el descriptor de contorno y la distancia local, δ, utilizada en el ATNLC también afectará.

En la Figura 6.10, tenemos un ejemplo recogido de nuestros experimentos[3] con la base de datos de contornos MPEG7b (comentada en la Sección 8.2.1.1), realizando una selección

[3]Los experimentos están ubicados en el Capítulo 8. Pero hemos creído conveniente poner aquí también estos resultados para facilitar la comprensión al lector.

de 100 puntos para cada contorno. Se han utilizado tres descriptores de contorno, la curvatura, BAS y los *shape contexts* (véase la Sección 2.2.11.1). Se puede ver que existe una similitud en la forma de las distribuciones de distancias (izquierda) y las correspondientes distribuciones de H (derecha) y la reducción de tripletas que incumplen la desigualdad triangular cuando sube la dimensionalidad. Con la curvatura, un 2,95 % de las tripletas, con BAS, un $7{,}25 \cdot 10^{-3}$ %, y finalmente con los *shape contexts*, un $7{,}07 \cdot 10^{-5}$ %.

En nuestros experimentos no tenemos tanta suerte como en [CVR87]. Ahora bien, son muy pocos los casos. Sí que se nota mucho con la curvatura, con casi un 3 %. Para los otros dos tipos de descriptores la cantidad es ridícula. Dada la naturaleza de AESA, los resultados no se van a ver afectados significativamente en la práctica para estos descriptores, que son en realidad los que necesitamos acelerar. Tal como se comenta en la Sección 2.2.11, las cadenas que describen cada punto con un elemento de muchas dimensiones son las que nos interesan, ya que son con las que se obtiene mejores resultados en recuperación de formas de la literatura por contener más información. Aunque claro, el que aumente la dimensionalidad intrínseca es bueno para el cumplimiento de la desigualdad triangular pero no lo es tanto para AESA, ya que degenera la búsqueda. Aun así, como veremos en el capítulo de experimentos los resultados son satisfactorios tanto en tiempo como en las tasas de acierto para los casos probados.

Si quisiéramos buscar más velocidad sacrificando la tasa de acierto y el cumplimiento de la desigualdad triangular, podríamos bajar la dimensionalidad intrínseca variando (dependiendo del caso) el descriptor de contorno utilizado y la distancia. Por ejemplo, en el caso de los *shape contexts* podríamos bajar el número de dimensiones para cada punto y utilizar la distancia euclídea en vez de χ^2.

Por otro lado, hasta ahora, aunque hemos considerado que tenemos una base de datos conocida, hemos supuesto que no sabemos la muestra que nos puede llegar a nuestro sistema para reconocer. En el caso de que las muestras que nos lleguen no difieran demasiado de nuestros prototipos, es decir, conozcamos el universo, podemos medir la holgura [VCR85] para que todos los prototipos cumplan la desigualdad triangular, y tener la seguridad de que no podamos incorrectamente. Eso sí, dependiendo de esta holgura, H, AESA iría más lento. La cota inferior quedaría como sigue:

$$d(s', y) \geq |d(s, s') - d(s, y)| + H.$$

Si conocemos el universo y nos interesa más la velocidad que la tasa de aciertos podemos utilizar la misma estrategia que se adopta en [VCR85, JM03]. La holgura, H, se utiliza de manera contraria a lo anteriormente explicado. Se busca ver cuánto puede desplazarse la distribución de H a la izquierda sin que perjudique a la tasa de aciertos (dada la naturaleza de nuestros datos y de AESA) y así podar mayor cantidad de prototipos. La cota inferior quedaría de la siguiente manera:

$$d(s', y) \geq |d(s, s') - d(s, y)| - H.$$

Además, podemos tener en cuenta lo siguiente. Cuanto más alta sea la dimensionalidad intrínseca, ϱ, muy probablemente podremos utilizar una holgura, H, con un valor más grande sin afectar a la tasa de aciertos.

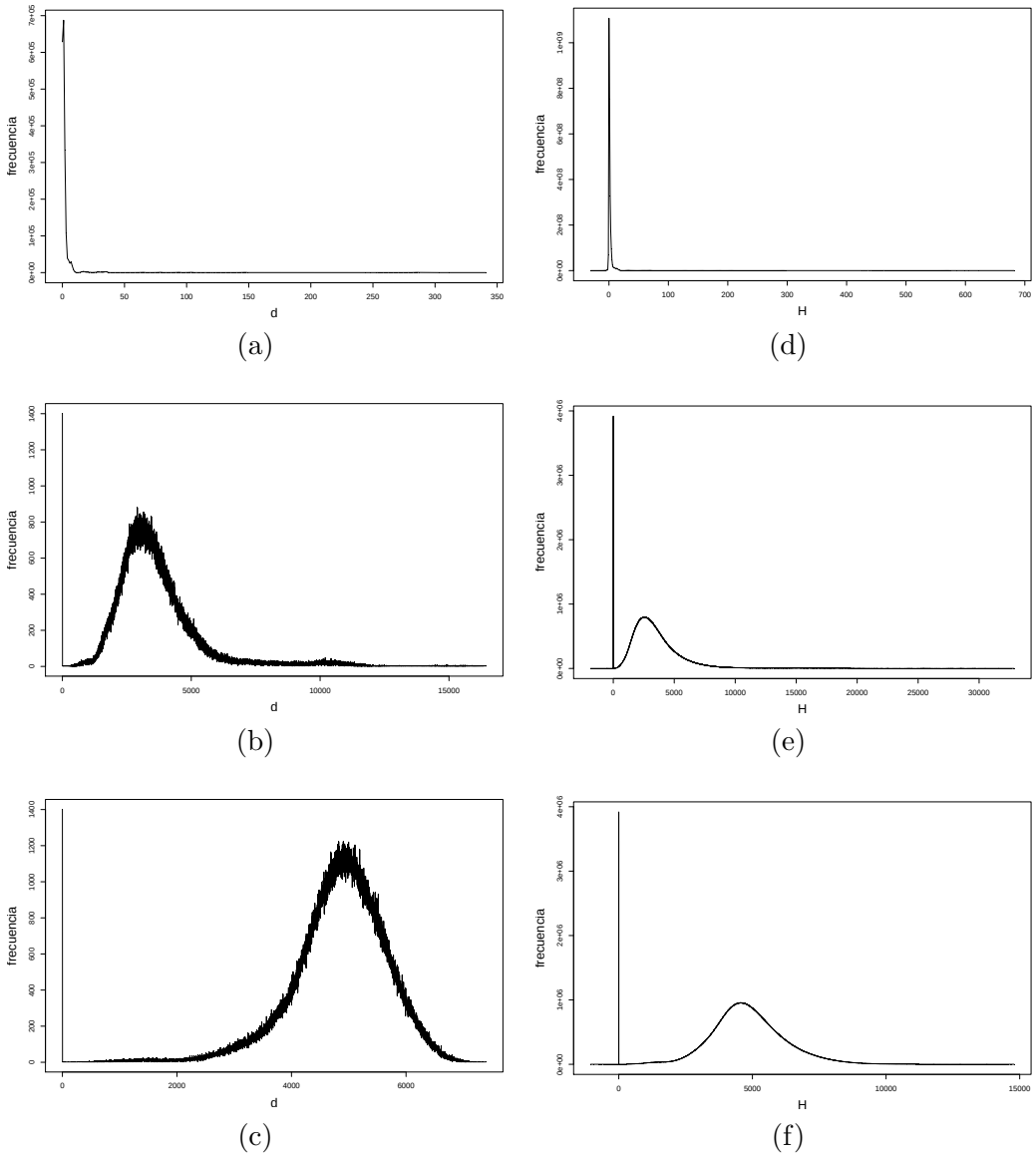


Figura 6.10: Los gráficos de la columna izquierda muestran la distribución de distancias de los descriptores de contorno, (a) curvatura, (b) BAS, y (c) shape contexts. Las dimensionalidades intrínsecas son, para cada caso, (a) $\varrho = 0{,}02566$, (b) $\varrho = 2{,}52256$ y (c) $\varrho = 17{,}59673$. Los gráficos de la derecha muestran la distribución de valores de H para los mismos descriptores que el gráfico de su izquierda. De arriba a abajo las tripletas que no cumplen la desigualdad triangular son, $2{,}95\,\%$, $7{,}25 \cdot 10^{-3}\,\%$ y $7{,}07 \cdot 10^{-5}\,\%$, respectivamente. El corpus utilizado ha sido el MPEG7b y se ha realizado una selección de 100 puntos.

6.4. Discusión

En este capítulo hemos propuesto una técnica aproximada del ATNLC con el heurístico APR, que reduce considerablemente el tiempo de computo, prácticamente la que nos proporcionaría el ATNL no cíclico, con un ligero empeoramiento de las tasas de reconocimiento, como veremos en el Capítulo 8. Además, hemos proporcionado un algoritmo óptimo para acelerar el, ya rápido de por sí, algoritmo divide y vencerás, propuesto en el Capítulo 5, para utilizarlo en tareas de clasificación y recuperación de formas.

Por otro lado, hemos propuesto el algoritmo LAESAEA, el cual acelera LAESA haciendo uso de la no continuación del cálculo de la distancia cuando vemos que no va a mejorar la cota externa (cuando no es un prototipo base). LAESAEA puede aplicarse también a otras distancias, siempre que tengamos una manera de parar el cálculo de estas. Como es el caso de la distancia de edición cíclica (con el algoritmo de ramificación y poda, véase la Sección 3.3.3), la propia distancia de edición, el ATNL u otras distancias basadas en programación dinámica. Finalmente, hemos visto que con los métodos basados en AESA no podemos llegar a una solución óptima pero que cuando la dimensionalidad intrínseca es relativamente alta, sí que nos acercamos a esta optimalidad. Por supuesto, esta afirmación puede aplicarse a otros contextos, no solo al caso del ATNLC.

En el Capítulo 8, veremos experimentalmente, qué mejoras proporcionan todas estas técnicas.

Capítulo 7

Modelos ocultos de Markov cíclicos

Porco Rosso: «*Hahaha... This isn't a Western, you can't hit me from here.* »
Hayao Miyazaki, *Pòrco Rósso* (1998).

Como se ha comentado en capítulos anteriores, los descriptores de formas, combinados con las técnicas de reconocimiento, deben ser invariantes a varias distorsiones, entre ellas, si estamos utilizando los contornos como descriptor, se encuentra la invarianza al punto inicial. La solución más adecuada para conseguir esta invarianza es tratar todo posible elemento inicial de la cadena, es decir, utilizar cadenas cíclicas. Una cadena cíclica modela el conjunto de todos los posibles desplazamientos cíclicos de una cadena convencional. Así, medir distancias entre dos cadenas cíclicas significa medir distancias entre todas los posibles desplazamientos cíclicos.

Ahora bien, la pregunta es la siguiente: ¿Cómo podemos entrenar o evaluar los modelos ocultos de Markov (MOMs) con cadenas cíclicas? Los MOMs se suelen utilizar para modelar cadenas con un orden temporal. En las cadenas cíclicas existe un orden relativo entre posiciones, pero no hay un «instante inicial». En este capítulo trataremos de resolver este problema.

7.1. Introducción

Tal como hemos comentado en capítulos anteriores, en la literatura existen diversos trabajos en los que se emplean los MOMs para reconocer formas bidimensionales a partir de sus contornos. Uno de los primeros trabajos [HK91] utiliza parámetros de un modelo autoregresivo derivados de la función radio-vector (Sección 2.2.2.2) del contorno, para entrenar modelos de Markov ergódicos. Se obtuvieron resultados prometedores. En [FMJ97], los contornos se describen con la derivada del código de cadena de 8 direcciones (Sección 2.2.3), utilizándose modelos ergódicos y modelos izquierda-derecha, dando mejores resultados esta última topología. Un descriptor con códigos de cadena similar se utilizó en [AYV00], pero con una topología circular, permitiendo de este modo invarianza al escalado y al punto inicial, según el autor. En [CL01] se utiliza el espectro de Fourier para describir el contorno y se propone un modelo de Markov y una reestimación específicos para manejar este

descriptor. Bicego *et al* [BM04] combina curvaturas con modelos ergódicos, seleccionando el número de estados del modelo mediante el *Bayesian Inference Criterion* [Sch78] sobre el agrupamiento de gaussianas de los valores de curvatura. Recientemente, este trabajo se mejora en [TGJ07], el cual utiliza la misma representación y topología, enfocándose en la reestimación de los modelos.

Más concretamente, si nos fijamos en cómo se consigue la invarianza al punto inicial (o el trabajar con cadenas cíclicas) en estos trabajos, veremos que aparecen las siguientes soluciones: la elección de referencia [HK91], características invariantes a la rotación [CL01], una topología circular [AYV00] y utilizar, sin más, un modelo ergódico [BM04], donde el entrenamiento resolverá el problema.

En primer lugar, hemos de tener en consideración que para modelar una clase de formas bidimensionales debemos utilizar un modelo de Markov de muchos estados para tener en cuenta todas las variaciones dentro de la clase. Muchos de los trabajos utilizan topologías ergódicas, lo cual tiene como consecuencia algunos problemas. En las topologías ergódicas es posible visitar estados más de una vez, habiendo pasado antes por otros estados, permitiendo transiciones en ambos sentidos entre pares de estados (véase la Figura 4.10a). Los modelos ergódicos no imponen restricciones severas de orden en la cadena de observaciones. Cuando la cadena de observaciones es temporal o existe un orden (como en el caso de los contornos), estas topologías no utilizan completamente la información secuencial de los datos y muchos de los estados se utilizan para explicar múltiples observaciones a lo largo del contorno. Esto hace que el reconocimiento sea un problema complejo. Además, el proceso de entrenamiento, en estos modelos es muy sensible a la inicialización y a la estimación local de parámetros.

Por esto, parecen más adecuadas las topologías izquierda-derecha. Estas topologías no permiten visitar estados que se hayan visitado previamente, a excepción de aquel en el que nos encontramos, obligando a que las transiciones cumplan: $a_{ij} = 0$ para $j < i$. En los modelos izquierda-a-derecha hay un estado inicial y los estados finales. Así, en el tiempo, la secuencia de estados recorrida se fuerza a empezar en el estado inicial (el de más a la izquierda, Figura 4.10b y Figura 4.10c) y hacer transiciones a estados posteriores (es decir, a la derecha) o al estado actual. De esta manera, se permite que la secuencia de estados represente el paso del tiempo. Cuando una secuencia de símbolos puede ser segmentada, todos los símbolos de un segmento son emitidos por el mismo estado, y segmentos consecutivos son asociados a estados consecutivos.

Aunque habitualmente estas topologías contienen más estados (pudiendo determinarse por las características que sobresalen del contorno), el número de transiciones es escaso, con lo que la complejidad de los algoritmos se reduce. Sin embargo, las cadenas cíclicas no tienen un punto inicial (ni tampoco final), los MOMs izquierda-derecha pueden parecer inapropiados para modelarlas.

En [AYV00] se propone una topología circular para modelar las cadenas cíclicas. La Figura 7.1a muestra esta topología, la cual puede ser vista como una modificación de la topología izquierda-derecha, donde el último estado emisor se conecta al primer estado emisor. Esta topología elimina la necesidad de definir un punto de inicio: la cadena cíclica puede segmentarse para asociar estados consecutivos a segmentos consecutivos en las cadenas cíclicas, pero no se hace ninguna asunción sobre cuál es el primer o últi-

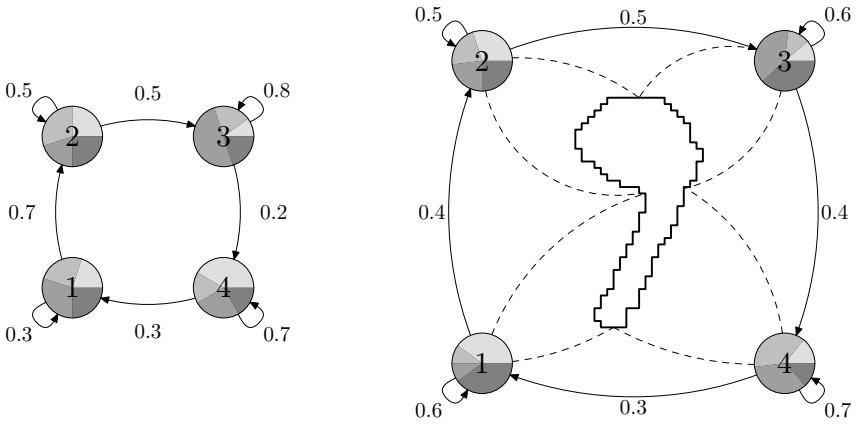


Figura 7.1: (a) MOM con la topología circular propuesto en [AYV00]. (b) El contorno de una forma es segmentado y cada segmento se asocia a un estado del MOM. Idealmente, cada estado es responsable de sólo un segmento.

mo segmento (véase la Figura 7.1b); por lo tanto, existe una analogía con las topologías izquierda-derecha. Sin embargo, hay un problema que rompe esta analogía: el modelo tiene problemas similares al ergódico (todos los estados pueden ser alcanzados desde cualquier estado y podemos acabar también en cualquiera de ellos). El camino óptimo de estados puede contener estados repetidos no consecutivos y, por tanto, un solo estado puede ser responsable de la emisión de varios segmentos no consecutivos de la cadena cíclica. Además, se puede producir un camino óptimo que no visite todos los estados al menos una vez.

7.2. Definición del problema de la ciclicidad

Para solucionar el problema de la ciclicidad de las cadenas podemos plantearlo de la siguiente manera[1]. El modelo de Markov ha generado una cadena que después ha sufrido un desplazamiento cíclico, pero no sabemos cuál. Es decir, un modelo λ ha generado una cadena $x = x_1 x_2 \dots x_m$ que ha sufrido la transformación $\rho^{k'}(x)$, pero no conocemos k'. Suponiendo que estos desplazamientos cíclicos son equiprobables, podemos tratar a x como a una cadena cíclica, $[x] = \{\rho^k(x) : 0 \leq k < m\}$.

[1]Aunque en la sección anterior hemos expuesto que una topología izquierda-derecha sería lo más adecuado, en un principio, el problema se puede tratar para cualquier tipo de topología.

Así, el problema de la evaluación mencionado en la Sección 4.3, puede resolverse con

$$\begin{aligned} P([x]|\lambda) &= \sum_{k=0}^{m-1} P(x|\lambda,k)P(k|\lambda) \\ &= \frac{1}{m}\sum_{k=0}^{m-1} P(x|\lambda,k) \\ &= \frac{1}{m}\sum_{k=0}^{m-1} P(\rho^k(x)|\lambda), \end{aligned} \tag{7.1}$$

es decir, deberemos calcular la probabilidad para todo posible desplazamiento cíclico y realizar la suma.

Del mismo modo, el problema de la decodificación (véase la Sección 4.3) puede solucionarse con

$$\begin{aligned} \hat{P}([x]|\lambda) &= \max_{0\le k\le m-1} \hat{P}(x|\lambda,k)P(k|\lambda) \\ &= \frac{1}{m}\max_{0\le k\le m-1} \hat{P}(\rho^k(x)|\lambda) = \max_{0\le k\le m-1} \hat{P}(\rho^k(x)|\lambda). \end{aligned} \tag{7.2}$$

La secuencia óptima de estados, $\hat{Q}$, se obtendrá pues del desplazamiento cíclico que obtenga la mayor puntuación de Viterbi.

En un principio, vamos a adoptar esta puntuación, $\hat{P}$, como estimación de la probabilidad real (7.1), ya que como se comentó en la Sección 4.3 es una muy buena aproximación. Además, como veremos en la Sección 7.5, es posible reducir el coste del cálculo de este procedimiento, y así acelerar el reconocimiento y el entrenamiento, con las topologías lineales.

7.3. Aprendizaje cíclico

Para resolver el problema del aprendizaje (véase la Sección 4.3) con la ciclicidad tenemos como objetivo estimar los valores del modelo de Markov, para maximizar la probabilidad de las cadenas cíclicas observadas. Es decir, nuestro objetivo consiste en maximizar:

$$P(X|\lambda) = \prod_{l=1}^{L} P([x]^{(l)}|\lambda) = \prod_{l=1}^{L} \frac{1}{m^{(l)}} \sum_{k=0}^{m^{(l)}-1} P(\rho^k(x)|\lambda), \tag{7.3}$$

siendo X, el conjunto de cadenas cíclicas, $X = \{[x]^{(1)}, [x]^{(2)}, \ldots, [x]^{(L)}\}$.

Para obtener los parámetros de λ que maximizan esta función, seguiremos un procedimiento iterativo. Primeramente, fijaremos unos valores arbitrarios para λ (típicamente equiprobables), después obtendremos nuevos valores de estos parámetros, en cada iteración, utilizando transformaciones crecientes, aplicando la desigualdad de Baum-Eagon [BE67]. Se garantiza que los nuevos valores estimados, incrementan el valor de la función objetivo con respecto a la iteración anterior y, por tanto, su convergencia.

Como sabemos que $\sum_{j=0}^{n} a_{ij} = 1, \forall\ 0 \leq i \leq n$ y que (7.3) es un polinomio con respecto a A. La nueva estimación, $\bar{a}_{ij}$, puede obtenerse con la desigualdad de Baum-Eagon [BE67]:

$$\begin{aligned} \bar{a}_{ij} &= \frac{\frac{\partial P(X|\lambda)}{\partial a_{ij}} a_{ij}}{\sum_{j=0}^{n} \frac{\partial P(X|\lambda)}{\partial a_{ij}} a_{ij}} \\ &= \frac{\sum_{l=1}^{L} \frac{\partial P([x]^{(l)}|\lambda)}{\partial a_{ij}} \frac{a_{ij}}{P([x]^{(l)}|\lambda)}}{\sum_{j=0}^{n} \sum_{l=1}^{L} \frac{\partial P([x]^{(l)}|\lambda)}{\partial a_{ij}} \frac{a_{ij}}{P([x]^{(l)}|\lambda)}}. \end{aligned} \tag{7.4}$$

Siguiendo sólo con el numerador:

$$\begin{aligned} \sum_{l=1}^{L} \frac{\partial P([x]^{(l)}|\lambda)}{\partial a_{ij}} \frac{a_{ij}}{P([x]^{(l)}|\lambda)} &= \sum_{l=1}^{L} \frac{1}{m^{(l)}} \sum_{k=0}^{m^{(l)}-1} \frac{\partial P(x^{(l)}|\lambda, k)}{\partial a_{ij}} \frac{a_{ij}}{\frac{1}{m^{(l)}} \sum_{k=0}^{m^{(l)}-1} P(x^{(l)}|\lambda, k)} \\ &= \sum_{l=1}^{L} \sum_{k=0}^{m^{(l)}-1} \frac{\partial P(\rho^k(x^{(l)})|\lambda)}{\partial a_{ij}} \frac{a_{ij}}{\sum_{k=0}^{m^{(l)}-1} P(\rho^k(x^{(l)})|\lambda)} \\ &= \sum_{l=1}^{L} \sum_{k=0}^{m^{(l)}-1} \frac{P(\rho^k(x^{(l)})|\lambda)}{\sum_{k=0}^{m^{(l)}-1} P(\rho^k(x^{(l)})|\lambda)} \left[\frac{\partial P(\rho^k(x^{(l)})|\lambda)}{\partial a_{ij}} \frac{a_{ij}}{P(\rho^k(x^{(l)})|\lambda)} \right]. \end{aligned}$$

Teniendo en cuenta que la parte que está entre corchetes es lo que finalmente se utiliza (obviamente, junto con el primer sumatorio) en el caso no cíclico, para obtener las fórmulas (4.4) y (4.5). Podemos ya concluir que en el caso cíclico:

$$\begin{aligned} \bar{a}_{ij} &= \frac{\sum_{l=1}^{L} \sum_{k=0}^{m^{(l)}-1} \text{Nº esperado de veces que se pasa de } S_i \text{ a } S_j \text{ con } \rho^k(x^{(l)})}{\sum_{l=1}^{L} \sum_{k=0}^{m^{(l)}-1} \text{Nº esperado de veces que se pasa por } S_i \text{ con } \rho^k(x^{(l)})} \\ &= \frac{\sum_{l=1}^{L} \sum_{k=0}^{m^{(l)}-1} \frac{1}{\sum_{k=0}^{m^{(l)}-1} P(\rho^k(x^{(l)})|\lambda)} \sum_{t=0}^{m^{(l)}-1} \alpha_t^{l_k}(i) a_{ij} b_j(\rho^k(x_{t+1}^{(l)})) \beta_{t+1}^{l_k}(j)}{\sum_{l=1}^{L} \sum_{k=0}^{m^{(l)}-1} \frac{1}{\sum_{k=0}^{m^{(l)}-1} P(\rho^k(x^{(l)})|\lambda)} \sum_{t=0}^{m^{(l)}-1} \alpha_t^{l_k}(i) \beta_{t+1}^{l_k}(j)}, \end{aligned} \tag{7.5}$$

siendo $\alpha_t^{l_k}$ y $\beta_t^{l_k}$, $\alpha_t(i)$ y $\beta_t(i)$ para $\rho^k(x^{(l)})$, respectivamente.

Siguiendo un razonamiento similar con $b_i(v_j)$ y sabiendo que $\sum_{j=0}^{w} b_i(v_j) = 1, \forall\ 1 \leq i \leq n$ y que (7.3) es un polinomio con respecto a B. Podemos llegar a:

$$\begin{aligned}\bar{b}_i(v_j) &= \frac{\sum_{l=1}^{L}\sum_{k=0}^{m^{(l)}-1} \text{Nº esperado de veces que se pasa por } S_i \text{ y se observa } v_j \text{ con } \rho^k(x^{(l)})}{\sum_{l=1}^{L}\sum_{k=0}^{m^{(l)}-1} \text{Nº esperado de veces que se pasa por } S_i \text{ con } \rho^k(x^{(l)})} \\ &= \frac{\sum_{l=1}^{L}\sum_{k=0}^{m^{(l)}-1} \frac{1}{\sum_{k=0}^{m^{(l)}-1} P(\rho^k(x^{(l)})|\lambda)} \sum_{\substack{t=1 \\ \forall \rho^k(x_t^{(l)})=v_j}}^{m^{(l)}-1} \alpha_t^{l_k}(i)\beta_t^{l_k}(i)}{\sum_{l=1}^{L}\sum_{k=0}^{m^{(l)}-1} \frac{1}{\sum_{k=0}^{m^{(l)}-1} P(\rho^k(x^{(l)})|\lambda)} \sum_{t=1}^{m^{(l)}-1} \alpha_t^{l_k}(i)\beta_t^{l_k}(i)}.\end{aligned} \tag{7.6}$$

Estamos ya en condiciones de presentar el procedimiento iterativo, el algoritmo de aprendizaje cíclico con Baum-Welch. Este se describe en la Figura 7.2. El coste del algoritmo en cada iteración es $O(Ln^2m^2)$.

De manera similar, puede realizarse un aprendizaje cíclico con Viterbi con la secuencia óptima de estados, $\hat{Q}$, de las cadenas cíclicas, con las siguientes fórmulas de reestimación:

$$\hat{a}_{ij} = \frac{\sum_{l=1}^{L} \text{Nº de veces que se pasa de } S_i \text{ a } S_j \text{ en } \hat{Q} \text{ con } \rho^k(x^{(l)})}{\sum_{l=1}^{L} \text{Nº de veces que se pasa por } S_i \text{ en } \hat{Q} \text{ con } \rho^k(x^{(l)})}, \tag{7.7}$$

$$\hat{b}_i(v_j) = \frac{\sum_{l=1}^{L} \text{Nº de veces que se pasa por } S_i \text{ y se observa } v_j \text{ en } \hat{Q} \text{ con } \rho^k(x^{(l)})}{\sum_{l=1}^{L} \text{Nº de veces que se pasa por } S_i \text{ en } \hat{Q} \text{ con } \rho^k(x^{(l)})}. \tag{7.8}$$

El algoritmo de aprendizaje iterativo con Viterbi se muestra en la Figura 7.3.
Siguiendo la línea de pensamiento de [JR90]:

Teorema 7.1 *El aprendizaje cíclico con Viterbi con cadenas cíclicas converge en el sentido de la convergencia global de Zangwill [Zan69].*

Demostración: Lo que se necesita mostrar es que $P([x]|\lambda)$ es una función creciente para este algoritmo. Sean Q^* y $\hat{Q}$ dos secuencias de estados óptimas tal que, $Q^* = \arg\max_Q P([x], Q|\lambda)$ y $\hat{Q} = \arg\max_Q P([x], Q|\hat{\lambda})$, entonces:

$$\begin{aligned}\max_Q P([x], Q|\hat{\lambda}) &\geq P([x], Q^*|\hat{\lambda}) \\ &= \max_{\lambda'} P([x], Q^*|\lambda') \\ &= \max_{\lambda'} \Big(\max_Q \big(\max_r P(\sigma^r(x), Q|\lambda') \big) \Big) \\ &\geq \max_Q P([x], Q|\lambda).\end{aligned} \tag{7.9}$$

La maximización sobre λ' en (7.9) puede ser reemplazada por el aprendizaje cíclico con Viterbi mencionado anteriormente. □

Figura 7.2: Algoritmo de aprendizaje con Baum-Welch cíclico.

Entrada: λ: modelo, $X = \{[x]^{(1)}, [x]^{(2)}, \ldots, [x]^{(L)}\}$: conjunto de cadenas cíclicas
Salida: $\bar{\lambda}$: modelo entrenado
var M_α, M_β, M_a, M_b: matriz $[0\,..\,n][1\,..\,m]$ de $\mathbb{R}$// suponiendo que todas las cadenas
// tienen tamaño m

```
Inicio
    λ̄ = λ
    mientras no haya convergencia hacer
        para l en 1 .. L hacer
            P_total = 0
            para k en 0 .. m^(l) − 1 hacer
                P_total += forward (λ̄, ρ^k(x^(l)), M_α)
            para k en 0 .. m^(l) − 1 hacer
                P = forward (λ̄, ρ^k(x^(l)), M_α)
                P = backward (λ̄, ρ^k(x^(l)), M_β)
                Calcular los valores esperados según (7.5) y (7.6) y
                    almacenarlos en M_a y M_b
        Reestimar λ̄ según (7.5) y (7.6)
    devolver λ̄
Fin
función forward(λ: modelo, x: cadena, M_α: matriz [0 .. n][1 .. m] de ℝ) : ℝ
Inicio
    Cálculo iterativo de P = Σ_{i=0}^{n} α_m(i) para x en la matriz M_α
    devolver P
Fin
función backward(λ: modelo, x: cadena, M_α: matriz [0 .. n][1 .. m] de ℝ) : ℝ
Inicio
    Cálculo iterativo de P = Σ_{i=0}^{n} β_0(i) para x en la matriz M_β
    devolver P
Fin
```

Figura 7.3: Algoritmo de aprendizaje con Viterbi cíclico.

Entrada: λ: modelo, $X = \{[x]^{(1)}, [x]^{(2)}, \ldots, [x]^{(L)}\}$: conjunto de cadenas cíclicas
Salida: $\hat{\lambda}$: modelo entrenado
var
$\hat{Q}, Q'$: vector $[1\,..\,m]$ de $\mathbb{N}$ // suponiendo que todas las cadenas
M_a, M_b: matriz $[0\,..\,n][1\,..\,m]$ de $\mathbb{R}$ // tienen tamaño m
Inicio
  $\hat{\lambda} = \lambda$
  mientras no haya convergencia **hacer**
    para l **en** $1\,..\,L$ **hacer**
      $\hat{P} = -\infty$
      para k **en** $0\,..\,m^{(l)} - 1$ **hacer**
        $P' =$ `viterbi` $(\hat{\lambda}, \rho^k(x^{(l)}), Q')$
        si $P' > \hat{P}$ **entonces**
          $\hat{P} = P'$
          $\hat{Q} = Q'$
      Calcular las frecuencias según (7.7) y (7.8) con $\hat{Q}$ y almacenarlas en M_a y M_b
    Reestimar $\hat{\lambda}$ según (7.7) y (7.8)
  devolver $\hat{\lambda}$
Fin

función `viterbi` (λ: modelo, x: cadena, $\hat{Q}$: vector $[1\,..\,m]$ *de* $\mathbb{N}$) : $\mathbb{R}$
Inicio
  Cálculo iterativo de $\hat{P} = \max_{i=0}^{n} \phi_m(i)$ y de $\hat{Q}$ para x
  devolver $\hat{P}$
Fin

7.4. Heurístico APR para la selección del punto inicial

El heurístico APR (mencionado en la Sección 6.1) también puede aplicarse para obtener una solución a los problemas planteados en la Sección 4.3 para el caso de las cadenas cíclicas.

El preproceso es el mismo, pudiéndose utilizar también el algoritmo de la Figura 6.2. Una vez tengamos elegido un buen punto inicial para los prototipos (el obtenido en el preproceso) podemos entrenar el modelo de cada categoría como si de cadenas (no cíclicas) se tratara. Para clasificar una muestra que nos llegue deberemos utilizar el ATNLCD (véase la Figura 6.3) para conseguir un buen punto inicial con el representante de cada categoría, es decir, de cada MOM, y así obtener las probabilidades (o las puntuaciones de Viterbi) de manera convencional.

Aunque como veremos en el capítulo de experimentos esta solución es peor que las anteriores en cuanto a tasas de clasificación, tanto el entrenamiento como el reconocimiento es mucho más rápido.

7.5. MOMs lineales cíclicos

En la Sección 7.1 hemos concluido que las topologías izquierda-derecha son las más adecuadas para modelar los contornos. Podemos ir más allá y aventurar que la topología lineal (véase la Figura 4.10b) quizás sea la mejor, ya que si nos vamos a una topología de Bakis (Figura 4.10c) o de más transiciones por estado, aumenta la complejidad, tal como ocurre con las topologías ergódicas. Además, si lo que queremos es modelar cadenas cíclicas, como es nuestro caso, los modelos lineales presentan características interesantes, que explotaremos en lo siguiente.

Para ello, utilizaremos una definición alternativa de los modelos de Markov, popularizada por el *Hidden Markov Model Toolkit* (HTK) [YOO+95]. En ella, además de aparecer un estado inicial no emisor (tal como teníamos nuestra definición hasta ahora), aparece otro estado no emisor final[2]. En la Figura 7.4 podemos ver un ejemplo. En la Figura 7.5 vemos también un ejemplo del cálculo iterativo de la puntuación de Viterbi con esta topología. Como se puede ver, existe un parecido entre el grafo del trellis [Rab89] y el grafo de alineamiento del ATNL (véase la Figura 4.2), y que por tanto, el coste del algoritmo es $O(nm)$ y no $O(n^2m)$. En lo sucesivo y tal como ocurre con el ATNL, a la secuencia óptima de estados, que produce el algoritmo de Viterbi, la llamaremos alineamiento óptimo. En este el «alineamiento» se produce entre un estado y un segmento de elementos de cadena contiguos a lo largo del contorno.

Para modelar las cadenas cíclicas de una manera adecuada, los MOMs deberían tener en cuenta que cualquier componente de la cadena se puede emitir por el primer estado emisor y cuando este ha sido elegido, la componente previa debe ser emitida por el último estado emisor. De este modo, podemos utilizar los MOMs lineales de una manera similar a como lo hacemos con las cadenas cíclicas. Un MOM lineal cíclico (MOMLC) puede verse

[2]Este nuevo estado varía ligeramente los algoritmos comentados hasta ahora. No mostraremos estas variaciones por ser de naturaleza sencilla [YOO+95].

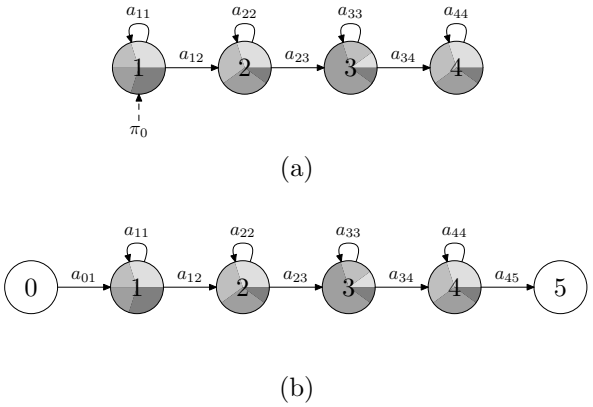


Figura 7.4: (a) Un MOM lineal. (b) Un MOM lineal utilizando la topología con dos estados no emisores, en el inicio y en el final.

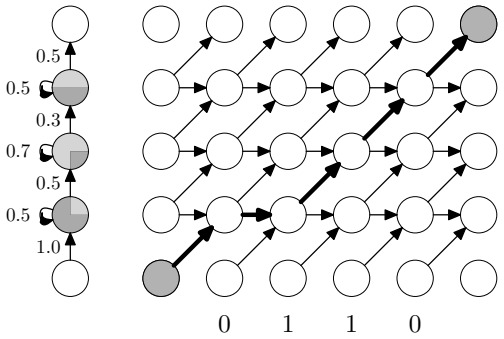


Figura 7.5: Trellis para un MOM lineal y una cadena de tamaño 4. El alineamiento óptimo se muestra con flechas más gruesas.

como el conjunto obtenido a partir de todos los desplazamientos cíclicos posibles de un MOM lineal (MOML):

Definición 7.2 *Sea $\lambda = (A, B)$ un MOML. Dado A, sea $\rho(A)$ la siguiente transformación:*

$$A = \begin{bmatrix} 1 & 0 & \dots & \dots & \dots & 0 \\ 0 & a_{11} & a_{12} & 0 & \dots & 0 \\ 0 & 0 & a_{22} & a_{23} & 0 & \dots & 0 \\ 0 & \dots & 0 & \ddots & \ddots & 0 & 0 \\ 0 & \dots & \dots & a_{nn} & a_{nn+1} & 0 \\ 0 & \dots & \dots & 0 & 0 \end{bmatrix},$$

$$\rho(A) = \begin{bmatrix} 1 & 0 & \dots & \dots & \dots & 0 \\ 0 & a_{22} & a_{23} & 0 & \dots & 0 \\ 0 & 0 & \ddots & \ddots & 0 & \dots & 0 \\ 0 & \dots & 0 & a_{nn} & a_{nn+1} & 0 & 0 \\ 0 & \dots & \dots & a_{11} & a_{12} & 0 \\ 0 & \dots & \dots & 0 & 0 \end{bmatrix}.$$

Sea $\rho(B) = \rho(b_1 \dots b_n) = b_2 \dots b_n b_1$ (donde b_i son filas de la matriz B). La composición de r desplazamientos cíclicos de λ se define como $\rho^r(\lambda) = (\rho^r(A), \rho^r(B))$. Dos MOMLs λ y λ' son cíclicamente equivalentes si $\lambda = \rho^r(\lambda')$, para algún r. La clase de equivalencia de λ es $[\lambda] = \{\rho^r(\lambda) : 0 \leq r < n\}$ y la podemos llamar MOML cíclico. *Cualquiera de los miembros de esta clase es un representante del MOML cíclico.*

En la Figura 7.6 podemos ver un ejemplo de MOML cíclico.
Entonces,

Definición 7.3 *La puntuación de Viterbi para una cadena cíclica $[x_1 x_2 \dots x_m]$ y un MOMLC $[\lambda]$ se define como*

$$\hat{P}([x]|[\lambda]) = \max_{0 \leq r < n} \big(\max_{0 \leq s < m} \hat{P}(\rho^s(x)|\rho^r(\lambda)) \big),$$

y esta puntuación tiene asociada un alineamiento óptimo.

La resolución de la ecuación de la Definición 7.3 tiene un alto coste computacional, pero el siguiente lema muestra que para calcular la puntuación de Viterbi sobre una cadena cíclica y un MOMLC, uno simplemente puede utilizar un representante del MOML y calcular la puntuación entre este MOML y la cadena cíclica.

Lema 7.4 $\hat{P}([x]|[\lambda]) = \hat{P}([x]|\lambda) = \max_{0 \leq s < m} \hat{P}(\rho^s(x)|\lambda)$.

Demostración: Consideremos un alineamiento óptimo Q_1 que representa el camino de máxima probabilidad entre λ y $\rho^{s_1}(x)$, para algún s_1, entonces, hay un alineamiento óptimo

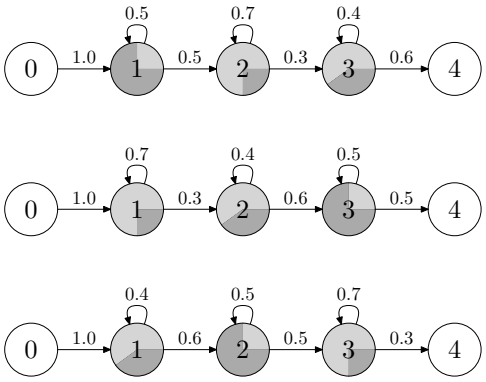


Figura 7.6: Un MOMLC representado por su conjunto de MOMLs.

Q_2 entre $\rho(\lambda)$ y $\rho^{s_2}(x)$, para algún s_2, tal que Q_2 representa exactamente los mismos componentes de la cadena emitidos por cada estado de la misma manera que Q_1. □

Este coste computacional, $O(m^2n)$, sigue siendo alto. Es por esto que proponemos un algoritmo más eficiente para evaluar esta puntuación. El algoritmo está inspirado también en el algoritmo de Maes para la distancia de edición cíclica [Mae90] (véase la Sección 3.3.2) y calcula la puntuación de Viterbi en tiempo $O(mn \log m)$. La puntuación se calcula en un trellis extendido donde la cadena original aparece concatenada con ella misma en el eje horizontal y los alineamientos deben empezar y terminar en nodos con el mismo color. Este inicio y final se corresponde con el tamaño de la cadena (véase la Figura 7.7). La eficiencia del algoritmo se basa, de nuevo, en la propiedad de no cruce de caminos: Sea Q_i el alineamiento óptimo que empieza en el nodo $(i, 0)$ y finaliza en el nodo $(m+i+1, n+1)$, en el trellis extendido y sea j, k, y l tres números enteros tal que $0 \leq j < k < l \leq m$; existe un camino óptimo que parte del nodo $(k, 0)$ y llega a $(k + m + 1, n + 1)$ y se encuentra entre Q_j and Q_l.

Esta propiedad nos lleva a un procedimiento recursivo divide y vencerás: cuando Q_j y Q_l se conocen, $Q_{(j+l)/2}$ se calcula solamente teniendo en cuenta los nodos que están entre Q_j y Q_l; al acabar, los alineamientos óptimos limitados por Q_j y $Q_{(j+l)/2}$ y los alineamientos óptimos limitados por $Q_{(j+l)/2}$ y Q_l se pueden calcular de manera recursiva. El procedimiento recursivo empieza después de calcular Q_0 (por medio de un algoritmo de Viterbi estándar) y Q_m, que es Q_0 desplazado m posiciones a la derecha. Cada llamada recursiva genera dos llamadas recursivas más y todos las llamadas en el mismo nivel de recursividad tienen un coste $O(mn)$; por lo tanto, el algoritmo se ejecuta en tiempo $O(mn \log m)$.

En la Figura 7.8 se detalla el algoritmo.

En principio, podríamos adoptar un enfoque simétrico realizando un desplazamiento cíclico sobre los estados de los MOMLs para obtener la misma puntuación de Viterbi. Esto significa «doblar» el MOM en el trellis extendido en vez de la cadena. Obtendríamos

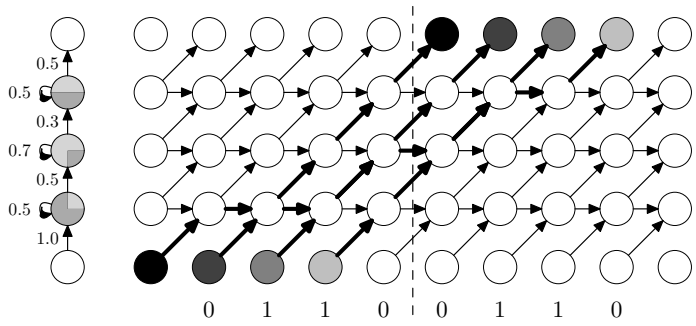


Figura 7.7: Trellis extendido para un MOM lineal y una cadena cíclica de 4 componentes. Los alineamientos óptimos para cada punto de inicio se muestran con flechas gruesas. Uno de ellos es el alineamiento óptimo de la cadena cíclica (el que tiene mayor puntuación).

Figura 7.8: Algoritmo divide y vencerás para calcular $P([x]|\lambda)$.

Entrada: x: cadena, λ: modelo
Salida: $\hat{p} : \mathbb{R}$
var V: vector $[0..m]$ de caminos de alineamiento
Inicio
  $p^* = \hat{P}(\rho^0(x)|\lambda)$
  Sea $V[0]$ el camino de alineamiento óptimo obtenido en el cálculo anterior
  Sea $V[m]$ igual a $V[0]$ desplazado m pasos a la derecha
  si $m > 1$ **entonces**
    $\hat{p} =$ `mín(`$\hat{p}$, `SiguientePaso(`$x \cdot x$, y, 0, m`))`
  devolver $\hat{p}$
Fin

función `SiguientePaso(`X: cadena, λ: modelo, $l : \mathbb{N}$, $r : \mathbb{N}$`)`:$\mathbb{R}$
Inicio
  $c = l + \lceil \frac{r-l}{2} \rceil$
  $p = \hat{P}(X_{c:c+m}, \lambda)$ conocidos $V[l]$ y $V[r]$
  si $l + 1 < c$ **entonces**
    $p =$ `mín(`p, `SiguientePaso(`X, y, l, c`))`
  si $c + 1 < r$ **entonces**
    $p =$ `mín(`p, `SiguientePaso(`X, y, c, r`))`
  devolver p
Fin

así un algoritmo de tiempo $O(mn \log n)$, que es mejor que $O(mn \log m)$ ya que $n \leq m$ (y, normalmente, $n \ll m$). Sin embargo, no puede hacerse de manera directa:

Lema 7.5 *En general, no es cierto que,* $\hat{P}([x]|[\lambda]) = \text{máx}_{0 \leq r < n} \hat{P}(x|\rho^r(\lambda))$.

Demostración: Utilicemos un contraejemplo. Sea $[x] = uvu$ una cadena cíclica con el alfabeto $\Sigma = \{u, v\}$. Sea $[\lambda]$ un MOMLC (discreto) con 2 estados emisores $a_{01} = 1$, $a_{11} = 0.5$, $a_{12} = 0.5$, $a_{22} = 0.5$, $a_{23} = 0.5$, $b_{01} = 1$, y $b_{12} = 1$. La definición de la puntuación de Viterbi del Lema 7.4 nos lleva a un valor de 0.125 (para la cadena $\rho^2(uvu) = uuv$). Si intentamos realizar un desplazamiento cíclico en el MOM lineal, tenemos dos posibles desplazamientos cíclicos. Ambas posibilidades nos dan 0 como puntuación de Viterbi. □

Sea $[\lambda]$ un MOMLC, sea $[x]$ una cadena cíclica y sea $\iota(\lambda)$ una operación que realiza un desplazamiento cíclico ($\rho(\lambda)$) e inserta una copia del primer estado emisor después del último estado, pero la probabilidad de la transición al siguiente estado tiene el valor de la transición a él mismo (véase la Figura 7.9). Entonces,

Teorema 7.6

$$\hat{P}([x]|[\lambda]) = \max_{0 \leq r < n} \Big(\text{máx} \left(\hat{P}(x|\rho^r(\lambda)), \hat{P}(x|\iota^r(\lambda)) \right) \Big).$$

Demostración: Cada alineamiento induce una segmentación en x. Todos las componentes de un segmento están alineadas con el mismo estado del MOMLC. Existe un problema cuando $x_{m-p}x_{m-p+1} \ldots x_m$ y $x_1x_2 \ldots x_q$, para algún $p, q \geq 0$, pertenece al mismo segmento de x. En este caso, el alineamiento óptimo no puede ser obtenido simplemente realizando un desplazamiento cíclico sobre λ, ya que x_m debe alinearse con el estado n y x_1 debe alinearse con el estado 1, es decir, nunca caen en el mismo segmento. El MOML $\iota^r(\lambda)$, formado insertando a $\rho^r(\lambda)$ el primer estado emisor después del último estado, permite alinear $x_{m-p}x_{m-p+1} \ldots x_m$ y $x_1x_2 \ldots x_q$ con el primer estado, ya que este estado también aparece al final de $\iota^r(\lambda)$. Por otro lado, supongamos que tenemos ahora el segmento completo al principio de la cadena, $p + q$ componentes, así, la primera auto-transición (transición al mismo estado) debe ser ejecutada $p + q - 1$ veces, pero si el segmento está en la situación explicada anteriormente, la primera auto-transición será utilizada sólo $p + q - 2$ veces. La transición al último estado no emisor proporciona esta transición extra necesaria. □

Corolario 7.7 $\hat{P}(x|\rho^r(\lambda))$, *para cada valor de* r, *puede obtenerse como subproducto del cálculo de* $\hat{P}(x|\iota^r(\lambda))$.

Demostración: El trellis que subyace para $\hat{P}(x|\rho^r(\lambda))$ es un subgrafo del que subyace para $\hat{P}(x|\iota^r(\lambda))$. El valor de $\hat{P}(x|\rho^r(\lambda))$ y $\hat{P}(x|\iota^r(\lambda))$, para cada r, puede obtenerse encontrando alineamientos óptimos en un trellis extendido, similar al mostrado en la Figura 7.7, pero en este caso «doblando» el MOML. Debería tenerse en cuenta que, a diferencia del algoritmo de Maes, el alinemiento óptimo que empieza en $(r, 0)$ puede finalizar tanto en el nodo $(r + n - 1, m)$ como el nodo $(r + n, m)$ y que el cálculo recursivo puede ser aplicado

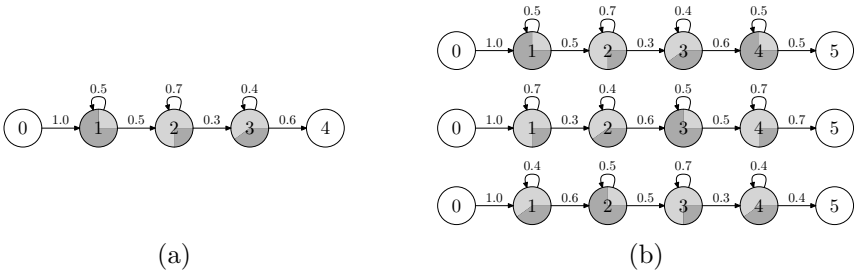


Figura 7.9: (a) Un MOMLC $[\lambda]$ representado por un MOML. (b) Los MOMLs correspondientes para la operación $\iota^r(\lambda)$, para $0 \leq r < n$ (donde $n = 3$). De arriba a abajo, $\iota^0(\lambda)$, $\iota^1(\lambda)$ and $\iota^2(\lambda)$

simplemente utilizando los alineamientos óptimos entre $\rho^r(\lambda)$ y x como un nuevo límite izquierdo o derecho. □

En la Figura 7.10 tenemos un ejemplo de este hecho.

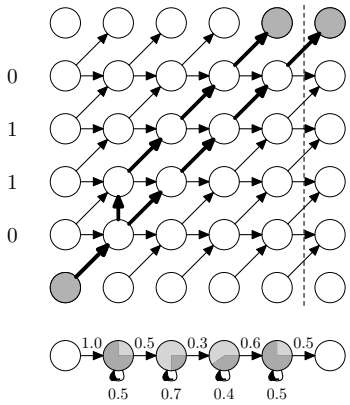


Figura 7.10: El mismo ejemplo de trellis que aparece en la Figura 7.5, pero en este caso, el modelo de Markov con topología lineal, aparece abajo con la operación $\iota^0(\lambda)$. Se puede ver que la operación $\hat{P}(x|\rho^0(\lambda))$ se puede calcular fácilmente a partir de este trellis.

Finalmente decir que, el algoritmo divide y vencerás propuesto puede ser utilizado, obviamente, para acelerar reconocimiento y el aprendizaje con Viterbi de cadenas cíclicas. Por desgracia, no puede ser extendido al procedimiento *forward* o *backward*, ya que no tenemos alineamientos óptimos en el trellis, y por tanto no podemos utilizar la propiedad de no cruce de caminos.

7.6. Discusión

En este capítulo hemos presentado diversas técnicas para el modelado con MOMs de cadenas cíclicas. Métodos de reconocimiento y de entrenamiento para modelos genéricos y finalmente para topologías lineales.

Debemos tener en cuenta que los métodos iterativos de entrenamiento obtienen un máximo local y por esta razón, una buena inicialización de los parámetros de los MOMs juega un rol muy importante. Para este propósito el heurístico APR comentado en el capítulo (para reconocer y entrenar) puede servir también para obtener una buena inicialización.

Por último comentar, que en el capítulo sólo hemos tratado con modelos de Markov discretos pero que en el caso de que necesitemos valores continuos y/o con elementos de varias dimensiones, la extensión de las fórmulas de reestimación es inmediata [Rab89].

En el Capítulo 8 veremos cómo se comportan estas técnicas con experimentos.

Capítulo 8

Experimentos

Arms Dealer: «*If you make money from war, you're scum. If you can't make money from bounty hunting, you're an idiot!*»
Hayao Miyazaki, *Pòrco Rósso* (1998).

En este capítulo se realiza un estudio experimental de las técnicas aportadas en la tesis en el marco del reconocimiento de formas bidimensionales con cadenas cíclicas. Presentamos experimentos para el ATNLC, las técnicas de aceleración y las extensiones cíclicas a los modelos ocultos de Markov.

8.1. Introducción

Los experimentos realizados se basan en tareas de clasificación y recuperación de formas. Para ello se han utilizado las siguientes bases de datos disponibles públicamente y muy frecuentemente usadas en la literatura para establecer comparaciones entre métodos: MPEG7 CE-Shape-1 (parte B), SQUID y Silhouette. También se ha utilizado una base de datos propia con teselas de mosaicos. Todas estas bases de datos se describirán con más detalle en la Sección 8.2.

En la Sección 8.3, mostraremos experimentos con el ATNLC (véase el Capítulo 6). Comprobaremos que el alineamiento cíclico ofrece resultados superiores al alineamiento convencional, como cabía esperar. Compararemos el ATNLC con otras técnicas de comparación de contornos relevantes de la literatura (véase la Sección 2.3), utilizando para ello descriptores de contorno destacados también (Sección 2.2).

En la Sección 8.4, veremos experimentalmente la mejora que producen las técnicas de aceleración (BBATNLC y métodos AESA) mencionadas en el Capítulo 6.

Por último, en la Sección 8.5, mostraremos experimentos de clasificación con los modelos ocultos de Markov con la decodificación y el entrenamiento cíclicos y otras metodologías para conseguir la invarianza al punto de inicio, todo ello presentado en el Capítulo 7.

8.2. Bases de datos

El objetivo de estos corpus es evaluar el comportamiento de técnicas de reconocimiento sobre formas bidimensionales bajo las condiciones de ruido, deformaciones no rígidas, cambios de escala y rotación. Todos los corpus constan de formas ya segmentadas definidas por sus contornos externos. Algunas bases de datos proporcionan ya el contorno codificado como secuencia de puntos y otras puede extraerse fácilmente al estar las imágenes binarizadas.

Para estas cadenas de puntos (en principio, cíclicas) se ha escogido un punto aleatorio de inicio. Después se ha realizado una selección de puntos fija, utilizando el método comentado en la Sección 2.2.4, para tres cantidades de puntos diferentes, 64, 100 y 128 puntos[1]. Hemos escogido estas cantidades porque son las más utilizadas en la literatura.

8.2.1. MPEG7 CE-Shape-1

Este corpus está dividido en tres partes [LLE00]. Cada parte plantea un problema y su correspondiente test para evaluar cómo se comporta el sistema reconocedor con este problema. Nosotros hemos utilizado solamente la parte B, que es la que presenta más dificultades y, precisamente por eso, la más utilizada en la literatura.

8.2.1.1. Parte B: Recuperación de formas basada en la similitud

El número total de formas en esta parte es de 1400: 70 clases con 20 formas por clase.

En el ámbito de la recuperación de formas es muy común realizar el experimento *bull-eye test* [Bob01], con esta parte B, para compararse con otros métodos de la literatura con un simple valor escalar. En este experimento las 1400 formas se utilizan como imagen de consulta y el número de formas similares que pertenecen a la misma clase se cuenta en las 40 primeras recuperadas. Debido a que el número máximo de aciertos para una consulta es de 20, el número total de posibles aciertos es 28000.

Hay que tener en cuenta que la parte B contiene clases con formas muy diferentes y formas de clases diferentes que son similares (véanse la Figura 8.1 y la Figura 8.2). Esto hace que sea muy complicado conseguir unas tasas de clasificación o de recuperación de formas con el número total de aciertos posible.

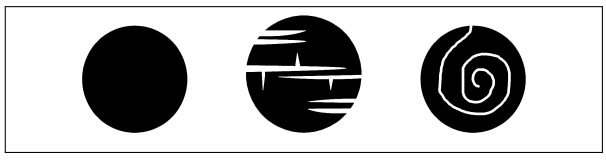


Figura 8.1: Imágenes de la parte B que pertenecen a la misma clase «device6» y presentan contornos muy diferentes.

[1]Hemos probado otros métodos de selección de puntos, como los descriptores de Fourier (véase la Sección 2.2.7) o métodos de aproximación poligonal (Sección 2.2.5), pero en general, este método nos ha dado los mejores resultados.

Figura 8.2: Imágenes de la parte B que pertenecen a diferentes clases y presentan contornos similares, de izquierda a derecha, las clases «*car*», «*truck*», «*fish*» y «*lmfish*».

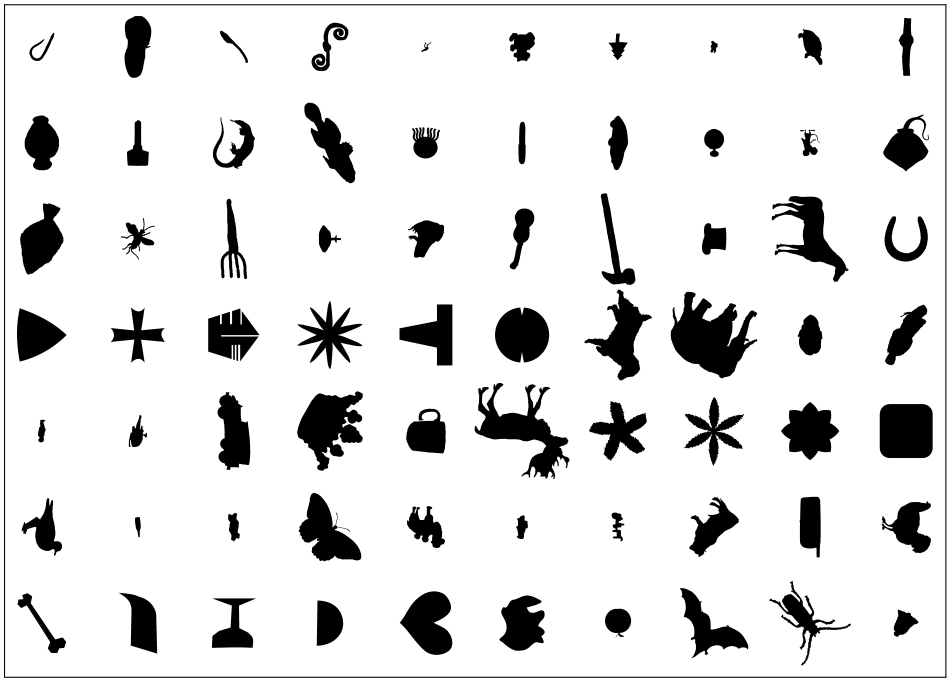


Figura 8.3: Algunas imágenes del corpus MPEG7 CE-Shape-1 (Parte B).

8.2.2. Silhouette

El corpus Silhouette [SCTK98], contiene 1070 contornos, divididos en 41 categorías que representan diferentes objetos (algunos ejemplos pueden verse en la Figura 8.4).

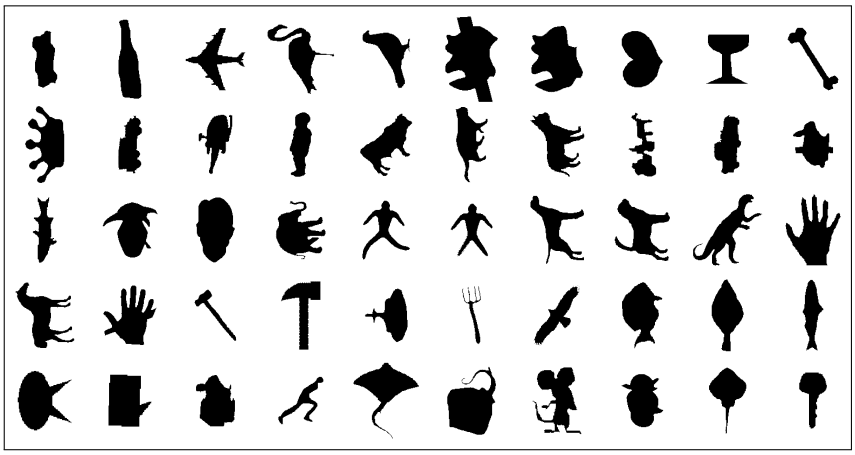


Figura 8.4: Algunas imágenes del corpus Silhouette.

8.2.3. SQUID

El corpus SQUID consiste en 1100 contornos de especies marinas (algunos ejemplos pueden verse en la Figura 8.5) y es utilizada como una aplicación de demostración de recuperación de formas en [MKA]. La base de datos original no está dividida en clases.

En [BCP05], se utiliza un etiquetado manual de 247 imágenes en 10 categorías semánticas[2]: seahorses (5 imágenes), seamoths (6), sharks (58), soles (52), tonguefishes (19), crustaceans (4), eels (26), u-eels (20), pipefishes (16), y rays (41).

8.2.4. Teselas de mosaicos

Un mosaico es una forma de arte decorativo donde pequeños azulejos (llamados teselas) son utilizados para crear una escena. Actualmente en la industria cerámica se montan estos mosaicos de manera manual, lo cual es muy trabajoso.

Están empezando a diseñarse métodos automáticos en los que las teselas circulan por una cinta transportadora. Con un sistema de clasificación se identifica la pieza y un robot la coloca en el lugar adecuado con la rotación adecuada basándose en un modelo. Es decir, el objetivo es, dado un contorno, encontrar la más parecida entre las teselas de un conjunto.

Este corpus consta de teselas de dos mosaicos (véase la Figura 8.6). No hemos tenido en cuenta los colores, con lo que el primer mosaico consta de 8 piezas diferentes y el segundo de 29 piezas diferentes.

[2]Que los autores han tenido la amabilidad de prestarnos.

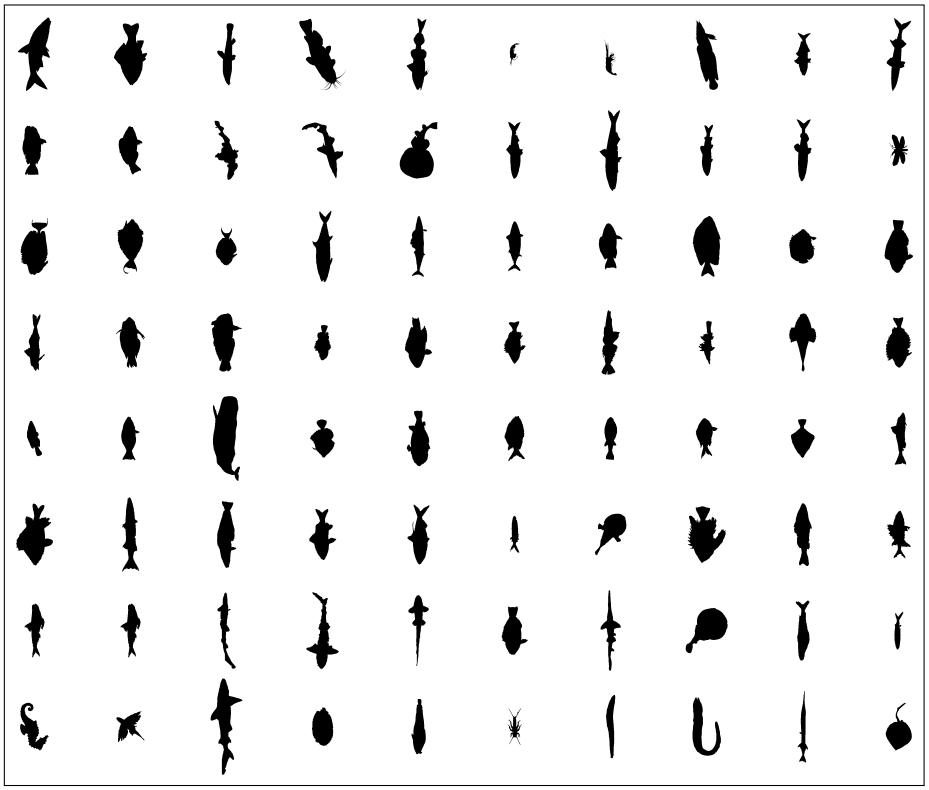
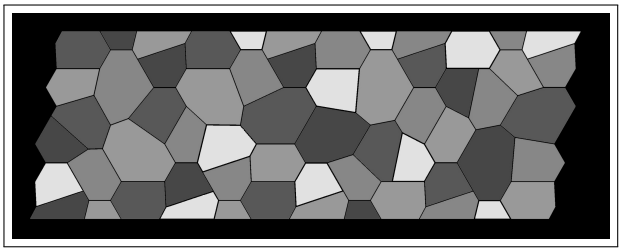
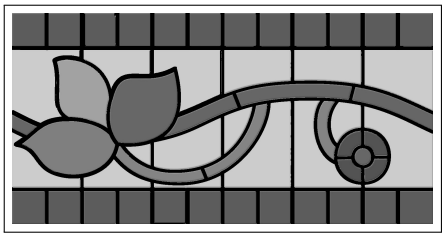


Figura 8.5: Algunas imágenes de la base de datos SQUID.



(a)



(b)

Figura 8.6: (a) Mosaico 1. (b) Mosaico 2.

8.3. Experimentos con el ATNLC

En esta sección veremos que el uso de cadenas cíclicas, y por tanto, del alineamiento temporal no lineal cíclico (ATNLC), es necesario para conseguir la invarianza al punto de inicio. Además evaluaremos el comportamiento del ATNLC utilizando diversos descriptores de contorno y lo compararemos con otras técnicas destacadas de la literatura.

La evaluación de los descriptores y métodos se ha llevado a cabo con:

- tasas de clasificación,
- gráficos *Precision-Recall* (PR), sólo para contornos de 128 puntos,
- *bull-eye test*, sólo en el caso del corpus MPEG7 CE-Shape-1 parte B,
- y la MAP (*Mean Average Precision*).

Todos ellos explicados en la Sección 2.5, o en el caso del *bull-eye test*, también en la Sección 8.2.1.1.

Cabe aquí explicar que todas estas medidas de evaluación son más altas cuanto mejor es el método. En el caso de las gráficas PR, pasa algo parecido, la técnica evaluada será mejor cuanto más cerca estén todos los puntos de la curva de una precisión del 100 %.

Los descriptores de contorno se han calculado a partir de la selección de puntos comentada en la Sección 8.2. A menos que se indique lo contrario, como distancia local (δ) para el alineamiento temporal no lineal (ATNL) se ha usado la distancia euclídea. A excepción del experimento con las teselas de mosaicos (Sección 8.3.4), en los experimentos de clasificación y recuperación de formas se ha utilizado la técnica *leaving-one-out* [DH73].

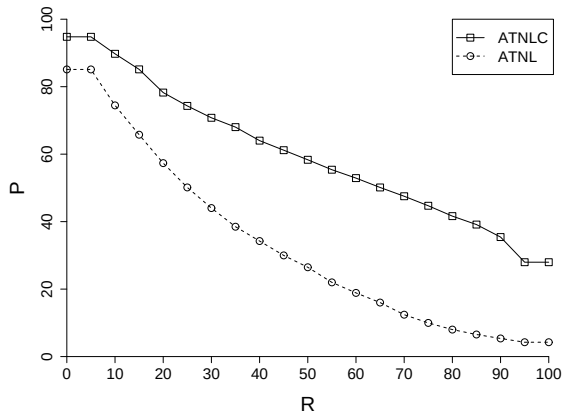
8.3.1. Comparación entre ATNL y ATNLC

Como experimento base tenemos una comparación entre el ATNL y el ATNLC con la curvatura como descriptor de contorno (véase la Sección 2.2.2.4). Las Figuras 8.7, 8.8 y 8.9 contienen los resultados.

Obviamente, el ATNLC funciona mucho mejor en todos los casos ya que, como hemos mencionado anteriormente, se ha escogido un punto arbitrario como inicio de las cadenas. Con este experimento se puede ver la importancia de escoger un buen punto de inicio para calcular la distancia.

Clasif.	**MPEG7B**		**Silhouette**		**SQUID**	
Puntos	**ATNL**	**ATNLC**	**ATNL**	**ATNLC**	**ATNL**	**ATNLC**
64	74.93	**87.29**	74.09	**87.93**	48.99	**60.32**
100	78.07	**90.50**	75.40	**91.77**	56.68	**66.80**
128	80.07	**92.57**	82.13	**93.08**	60.32	**72.06**

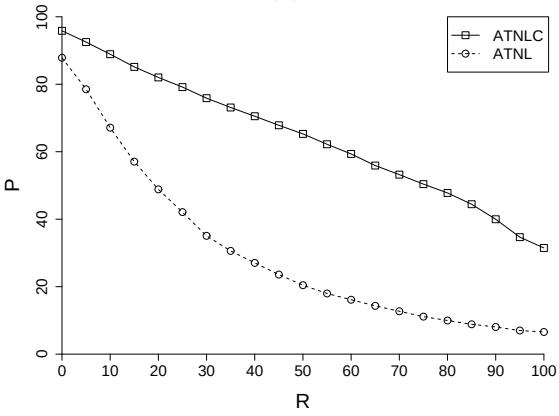
Figura 8.7: Tasas de clasificación (porcentaje de aciertos) para la comparativa entre ATNL y ATNLC, utilizando la curvatura como descriptor de contorno.



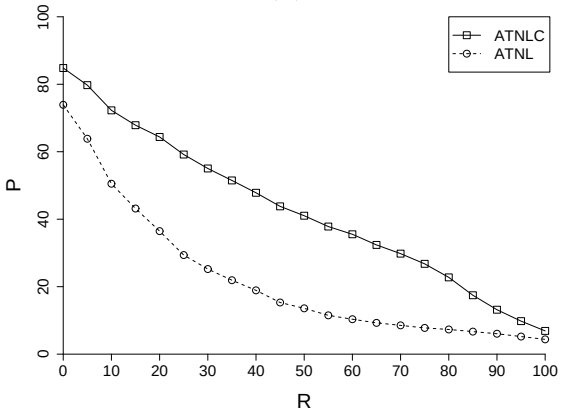
(a)

Bull-eye	MPEG7B	
Puntos	**ATNL**	**ATNLC**
64	41.08	**60.33**
100	40.38	**63.84**
128	41.11	**67.13**

(b)



(c)



(d)

Figura 8.8: Gráficos PR y test bull-eye para la comparativa entre ATNL y ATNLC, utilizando la curvatura como descriptor de contorno. (a) Gráfico PR para MPEG7B, (b) Bull-eye, (c) gráfico PR para Silhouette y (d) gráfico PR para SQUID. Todos los gráficos PR son de contornos de 128 puntos.

MAP	MPEG7B		Silhouette		SQUID	
Puntos	**ATNL**	**ATNLC**	**ATNL**	**ATNLC**	**ATNL**	**ATNLC**
64	29.78	**50.30**	25.71	**53.57**	14.31	**24.46**
100	30.52	**55.75**	28.24	**60.44**	17.13	**34.76**
128	31.67	**59.59**	28.53	**64.10**	20.57	**41.60**

Figura 8.9: MAP de la comparativa entre ATNL y ATNLC, utilizando la curvatura como descriptor de contorno.

8.3.2. Invarianza al punto inicial

En la Sección 2.4 se comentan varias formas de solucionar el problema de la invarianza al punto inicial, sin tener en cuenta todos los posibles desplazamientos cíclicos (es decir, el alineamiento cíclico). En la Sección 6.1 hemos aportado también un heurístico (alineamiento con patrón de referencia, APR) con este mismo fin, aunque sólo utilizable en el caso de que conozcamos las categorías.

Para comparar nuestro heurístico y el ATNLC con las soluciones de la literatura, utilizaremos los dos métodos más representativos: la elección del punto inicial con los descriptores de Fourier [BCP05] (FDs) y el espectro de Fourier [ZL05], para las características invariantes a la rotación. Para solucionar el problema de la ambigüedad que suele aparecer en los métodos de la elección de la rotación de referencia (véase la Sección 6.1), en vez de utilizar un heurístico, hemos calculado los dos ATNL y nos hemos quedado con la menor distancia.

En las Figuras 8.10, 8.11 y 8.50 tenemos los resultados de la comparación. Los resultados con el heurístico APR sólo aparecen en el experimento de clasificación, ya que en la recuperación de formas no se dispone de información sobre las categorías (véase la Sección 6.1). Hemos utilizado la curvatura como descriptor de contorno. Para medir la distancia entre los espectros de Fourier hemos utilizado la distancia euclídea (tal como se hace en [ZL05]). Para las rotaciones de referencia con los FDs hemos hecho, evidentemente, uso del ATNL.

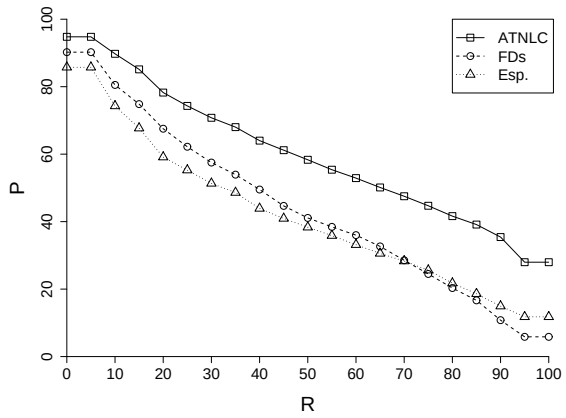
En general, se puede ver que el ATNLC es superior a los otros métodos y que el heurístico APR ofrece unos resultados muy buenos frente a las otras técnicas en la tarea de clasificación. Con el APR perdemos un poco de tasa de acierto frente al ATNLC, pero tiene un coste computacional menor, prácticamente $O(mn)$.

Clasif.	**MPEG7B**		**Silhouette**		**SQUID**	
Puntos	**Esp.**	**FDs**	**Esp.**	**FDs**	**Esp.**	**FDs**
64	77.71	76.14	70.81	76.24	48.18	50.61
100	79.57	85.29	73.81	85.31	54.25	60.73
128	81.36	86.93	73.25	89.43	57.89	67.21
	APR	**ATNLC**	**APR**	**ATNLC**	**APR**	**ATNLC**
64	83.57	**87.29**	87.00	**87.93**	57.49	**60.32**
100	87.21	**90.50**	89.62	**91.77**	**66.80**	**66.80**
128	89.21	**92.57**	92.42	**93.08**	**72.87**	72.06

Figura 8.10: Tasas de clasificación (porcentaje de aciertos) para la comparativa entre el espectro de Fourier (Esp.), la elección del punto inicial con los FDs, el heurístico ARP y el ATNLC. La curvatura se usa como descriptor.

8.3.3. Algunos descriptores de contorno

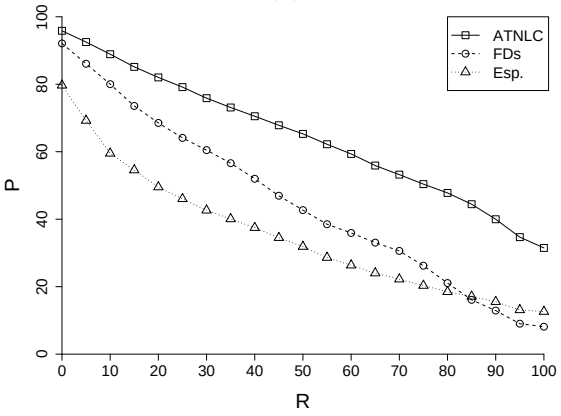
Hasta ahora sólo hemos hecho experimentos con la curvatura como descriptor de contorno, en esta sección comparamos la distancia al centroide (véase la Sección 2.2.2.2), la



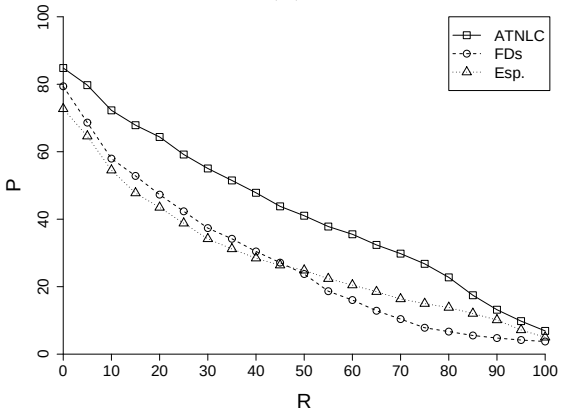
(a)

Bull-eye	MPEG7B		
Puntos	Esp.	FDs	ATNLC
64	52.27	44.01	**60.33**
100	51.76	48.59	**63.84**
128	50.43	51.40	**67.13**

(b)



(c)



(d)

Figura 8.11: Gráficos PR y test bull-eye para la comparativa entre el espectro de Fourier (Esp.), la elección del punto inicial con los FDs y el ATNLC usando la curvatura como descriptor. (a) Gráfico PR para MPEG7B, (b) Bull-eye, (c) gráfico PR para Silhouette y (d) gráfico PR para SQUID. Todos los gráficos PR son de contornos de 128 puntos.

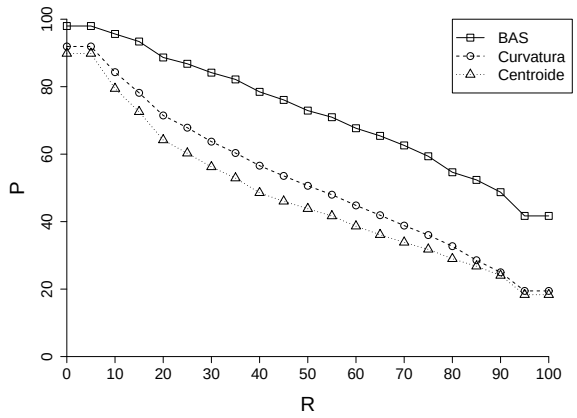
MAP	MPEG7B			Silhouette			SQUID		
Puntos	Esp.	FDs	ATNLC	Esp.	FDs	ATNLC	Esp.	FDs	ATNLC
64	40.40	33.52	**50.30**	35.24	32.92	**53.57**	20.98	19.49	**24.46**
100	40.77	40.63	**55.75**	34.82	40.12	**60.44**	23.73	22.74	**34.76**
128	40.78	43.61	**59.59**	34.18	44.78	**64.10**	27.09	26.44	**41.60**

Figura 8.12: MAP para la comparativa entre el espectro de Fourier (Esp.), la elección del punto inicial con los FDs y el ATNLC usando la curvatura como descriptor.

curvatura y el descriptor BAS (Sección 2.2.11.2), utilizando como medida de disimilitud el ATNLC. Nótese que los dos primeros descriptores otorgan un valor para cada punto, sin embargo, BAS otorga 4 valores a cada punto, es decir, cada punto tiene un valor en un espacio de 4 dimensiones. Esto produce que el punto quede mejor descrito (y por tanto el contorno) y los resultados de clasificación y recuperación de formas son mejores. Pero esto no es gratis: la comparación entre puntos (distancia local) es más costosa y repercute en el tiempo de computo del ATNLC (en la Sección 8.4 veremos de cuánto tiempo estamos hablando). Los resultados se muestran en las Figuras 8.13, 8.14 y 8.15.

Clasif.	**MPEG7b**			**Silhouette**			**SQUID**		
Puntos	**Cen.**	**Cur.**	**BAS**	**Cen.**	**Cur.**	**BAS**	**Cen.**	**Cur.**	**BAS**
64	86.93	87.29	**96.00**	92.70	87.93	**97.01**	55.06	60.32	**76.52**
100	87.14	90.50	**96.86**	93.17	91.77	**96.91**	54.66	66.80	**78.54**
128	87.57	92.57	**96.86**	92.42	93.08	**97.01**	59.92	72.06	**80.16**

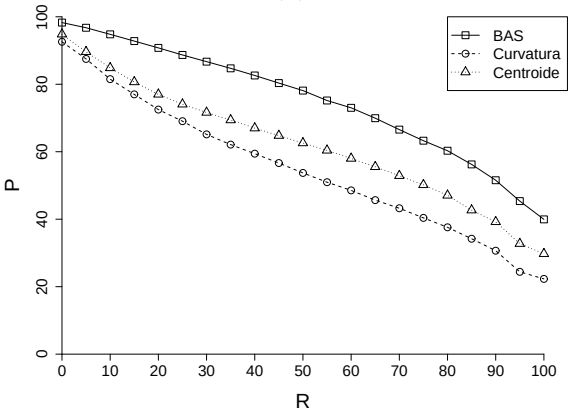
Figura 8.13: Tasas de clasificación (porcentaje de aciertos) para la comparativa entre la distancia al centroide, la curvatura y el BAS, utilizando como medida de disimilitud el ATNLC.



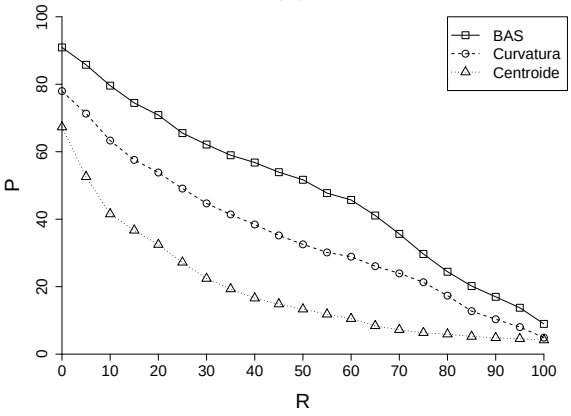
(a)

Bull-eye	MPEG7B		
Puntos	Cen.	Cur.	BAS
64	56.12	60.33	**76.58**
100	56.27	63.84	**77.43**
128	56.26	67.13	**77.74**

(b)



(c)



(d)

Figura 8.14: Gráficos PR y test bull-eye para la comparativa de la distancia al centroide, curvatura y BAS, utilizando como medida de disimilitud el ATNLC. (a) Gráfico PR para MPEG7B, (b) Bull-eye, (c) gráfico PR para Silhouette y (d) gráfico PR para SQUID. Todos los gráficos PR son de contornos de 128 puntos.

MAP	MPEG7B			Silhouette			SQUID		
Puntos	Cen.	Cur.	BAS	Cen.	Cur.	BAS	Cen.	Cur.	BAS
64	47.72	50.30	**70.81**	63.64	53.57	**73.46**	18.68	24.46	**46.75**
100	47.90	55.75	**71.88**	63.70	60.44	**74.53**	19.25	34.76	**47.61**
128	47.83	59.59	**72.26**	63.55	64.10	**74.88**	19.25	41.60	**48.05**

Figura 8.15: MAP para la comparativa entre la distancia al centroide, la curvatura y el BAS, utilizando como medida de disimilitud el ATNLC.

8.3.4. Comparación entre el CSS y el ATNLC

Para compararnos con el CSS, el método utilizado por el estándar MPEG7 [Bob01, MB03] para representar y comparar contornos (Sección 2.2.8), hemos realizado dos experimentos.

El primer experimento es una tarea de clasificación en la que hemos utilizado el corpus de las teselas de mosaicos (Sección 8.2.4). Hemos mezclado los dos mosaicos en uno sólo, y lo hemos duplicado para crear un conjunto de test y otro de clasificación. Este corpus es especialmente complicado para el CSS (como se puede ver en la Figura 8.16) debido a que algunas teselas, a pesar de ser relativamente complejas, están compuestas de curvas totalmente convexas.

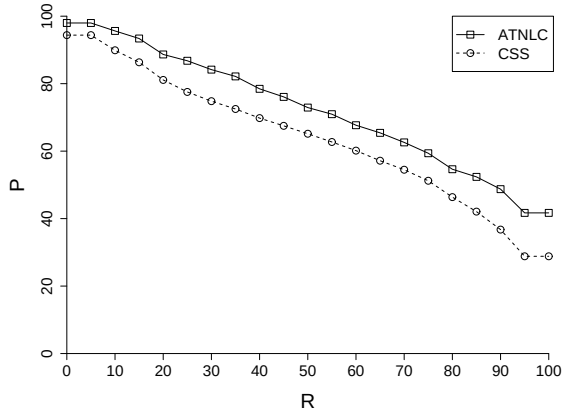
En el segundo experimento comparamos el CSS con el ATNLC y BAS (véanse las Figuras 8.17, 8.18 y 8.19), viendo que el CSS es superado claramente.

Clasif.	**Teselas**	
Puntos	**CSS**	**ATNLC**
64	61.97	**97.89**
100	60.56	**100.00**
128	59.15	**100.00**

Figura 8.16: Tasas de clasificación (porcentaje de aciertos) para la comparativa entre CSS y ATNLC con el corpus de las teselas de mosaicos. El descriptor utilizado por el ATNLC es BAS.

Clasif.	**MPEG7B**		**Silhouette**		**SQUID**	
Puntos	**CSS**	**ATNLC**	**CSS**	**ATNLC**	**CSS**	**ATNLC**
64	89.93	**96.00**	89.71	**97.01**	66.40	**76.52**
100	91.00	**96.86**	90.93	**96.91**	65.59	**78.54**
128	91.00	**96.86**	91.21	**97.01**	64.37	**80.16**

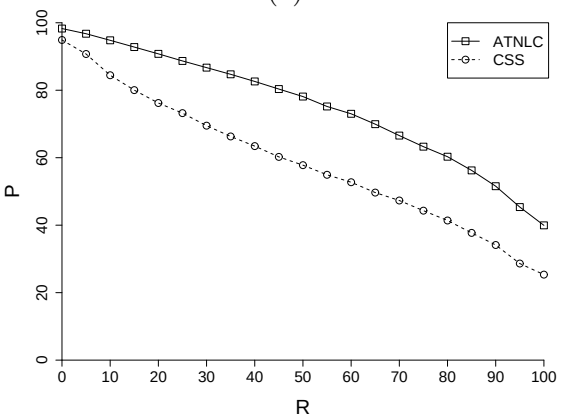
Figura 8.17: Tasas de clasificación (porcentaje de aciertos) para la comparativa entre CSS y ATNLC. El descriptor utilizado por el ATNLC es BAS.



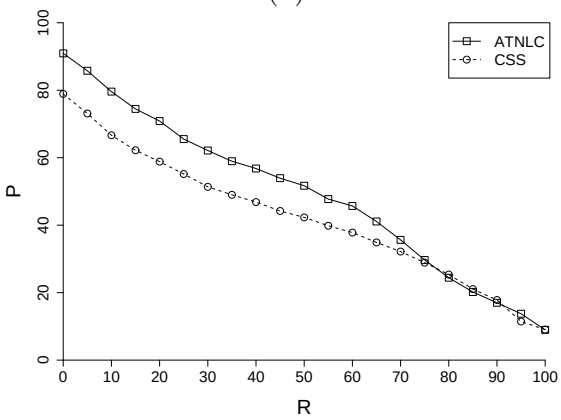
(a)

Bull-eye	MPEG7B	
Puntos	CSS	ATNLC
64	71.93	**76.58**
100	73.45	**77.43**
128	73.54	**77.74**

(b)



(c)



(d)

Figura 8.18: Gráficos PR y test bull-eye para la comparativa entre CSS y ATNLC. El descriptor utilizado por el ATNLC es BAS. (a) Gráfico PR para MPEG7B, (b) Bull-eye, (c) gráfico PR para Silhouette y (d) gráfico PR para SQUID. Todos los gráficos PR son de contornos de 128 puntos.

MAP	MPEG7B		Silhouette		SQUID	
Puntos	CSS	ATNLC	CSS	ATNLC	CSS	ATNLC
64	61.30	**70.81**	56.53	**73.46**	41.39	**46.75**
100	63.36	**71.88**	57.33	**74.53**	40.49	**47.61**
128	63.48	**72.26**	57.89	**74.88**	40.94	**48.05**

Figura 8.19: MAP para la comparativa entre CSS y ATNLC. El descriptor utilizado por el ATNLC es BAS.

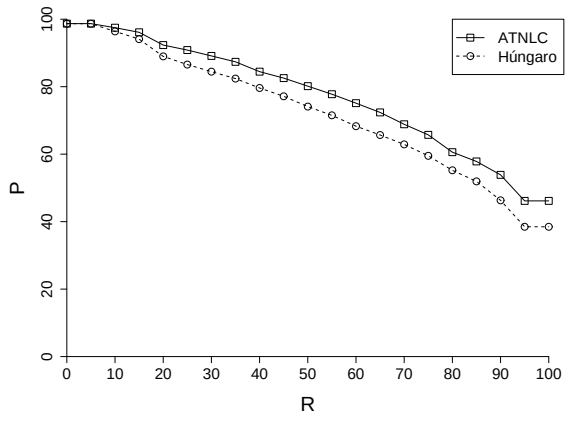
8.3.5. Comparación con el método húngaro y los *shape contexts*

Como hemos comentado en la Sección 8.3.3, el descriptor BAS utiliza 4 valores por cada punto. En esta sección experimentamos con descriptores todavía más pesados, los *shape contexts* (véase la Sección 2.2.11.1) y TAR (Sección 2.2.11.3).

En [BMP02] el reconocimiento con los *shape contexts* se realiza utilizando como medida de disimilitud el método húngaro (Sección 8.3.5) junto con χ^2 como función de coste. En las Figuras 8.20, 8.21 y 8.22) tenemos los resultados experimentales de la comparativa entre el método húngaro y el ATNLC (con χ^2 como distancia local, δ) sobre los *shape contexts*. Aunque con el método húngaro se obtienen muy buenos resultados, como se puede observar, el ATNLC los supera. El método húngaro es quizás el método más potente para buscar correspondencia entre puntos, sin embargo, en el caso de los contornos, peca de no utilizar la información de orden de las cadenas, cosa que el ATNLC sí hace. Además, el método húngaro tiene un coste computacional muy alto [BMP02], $O(n^3)$, en comparación con el ATNLC, $O(n^2 \log n)$, suponiendo que ambas cadenas tienen la misma longitud.

Clasif.	**MPEG7B**		**Silhouette**		**SQUID**	
Puntos	**Húngaro**	**ATNLC**	**Húngaro**	**ATNLC**	**Húngaro**	**ATNLC**
64	97.29	**97.86**	98.13	**98.60**	**78.14**	**78.14**
100	97.71	**98.29**	98.41	**98.78**	79.35	**81.38**
128	97.93	**98.07**	98.41	**98.69**	78.14	**80.57**

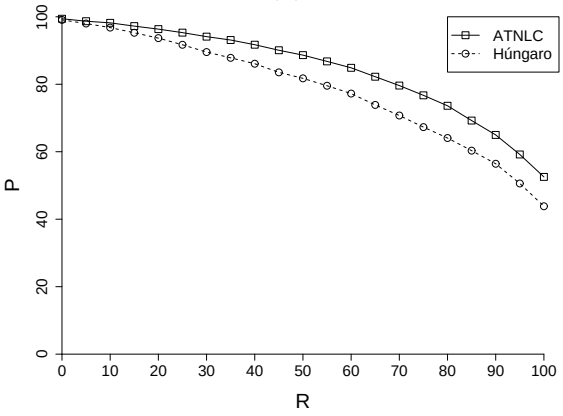
Figura 8.20: Tasas de clasificación (porcentaje de aciertos) para la comparativa entre el algoritmo Húngaro y ATNLC. El descriptor utilizado por ambos son los *shape contexts*.



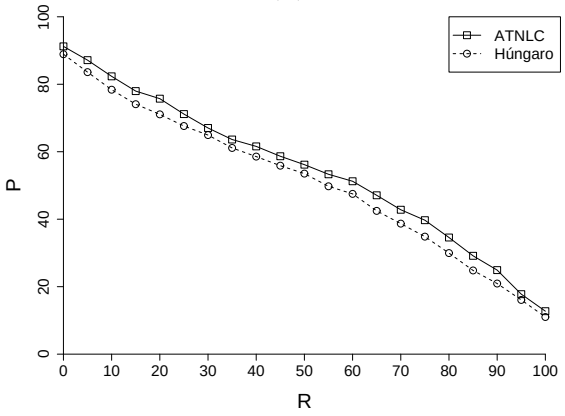
(a)

Bull-eye	MPEG7B	
Puntos	Húngaro	ATNLC
64	76.90	**80.16**
100	78.12	**81.61**
128	78.36	**82.08**

(b)



(c)



(d)

Figura 8.21: Gráficos PR y test bull-eye para la comparativa entre el algoritmo Húngaro y ATNLC. El descriptor utilizado por ambos son los *shape contexts*. (a) Gráfico PR para MPEG7B, (b) Bull-eye, (c) gráfico PR para Silhouette y (d) gráfico PR para SQUID. Todos los gráficos PR son de contornos de 128 puntos.

MAP	MPEG7B		Silhouette		SQUID	
Puntos	Húngaro	ATNLC	Húngaro	ATNLC	Húngaro	ATNLC
64	70.72	**75.73**	77.68	**83.01**	48.42	**51.07**
100	72.17	**77.18**	78.15	**84.00**	49.97	**53.30**
128	72.43	**77.49**	78.41	**84.51**	49.93	**53.40**

Figura 8.22: MAP para la comparativa entre el algoritmo Húngaro y ATNLC. El descriptor utilizado por ambos son los *shape contexts*.

8.3.6. Comparación con TAR

Hemos querido realizar finalmente, una comparativa, entre BAS, los *shape contexts* y TAR [ARKF07] (el método con el que se han conseguido los mejores resultados en el *bull-eye-test*), utilizando como medida de disimilitud el ATNLC. Los resultados se muestran en las Figuras 8.23, 8.24 y 8.25). Nótese que el resultado de TAR, con 128 puntos en el *bull-eye-test*, no se corresponde[3] con el mencionado en [ARKF07], donde se consigue un 85.03.

Hay que comentar también que el descriptor TAR, presenta unas dimensiones que están en función del tamaño de las cadenas [ARKF07], esto no ocurre en los *shape contexts* que el número de dimensiones es una constante k (en [BMP02] $k = 60$). Es por esto que el coste computacional del TAR es superior.

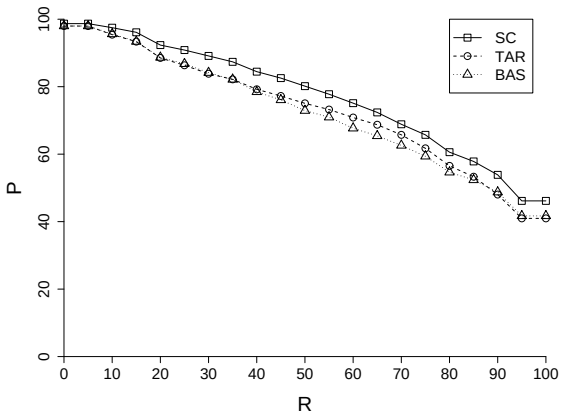
Como puede observarse los descriptores de puntos con información global (*shape contexts* y TAR) ofrecen muy buenos resultados pero tienen también un coste elevado.

En la siguiente sección mostramos también estos últimos experimentos para otras ponderaciones del ATNLC (véase la Sección 5.3). En general, otras ponderaciones no ofrecen mejores resultados.

Clasif.	**MPEG7B**			**Silhouette**			**SQUID**		
Puntos	**BAS**	**SC**	**TAR**	**BAS**	**SC**	**TAR**	**BAS**	**SC**	**TAR**
64	96.00	**97.86**	96.79	97.01	**98.60**	95.04	76.52	**78.14**	76.52
100	96.86	**98.29**	96.93	96.91	**98.78**	95.23	78.54	**81.38**	76.11
128	96.86	**98.07**	96.86	97.01	**98.69**	95.32	80.16	**80.57**	76.52

Figura 8.23: Tasas de clasificación (porcentaje de aciertos) para la comparativa entre los descriptores BAS, *shape contexts* y TAR, utilizando ATNLC como medida de disimilitud.

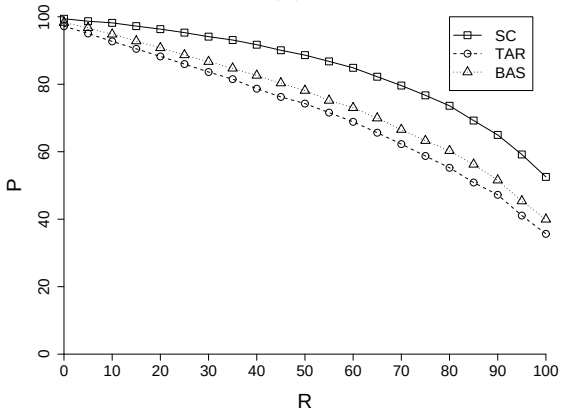
[3]Creemos que nuestra implementación sigue al pie de la letra lo expuesto en [ARKF07]. Nos hemos puesto en contacto con el autor pero las soluciones que nos ha planteado no han mejorado estos resultados.



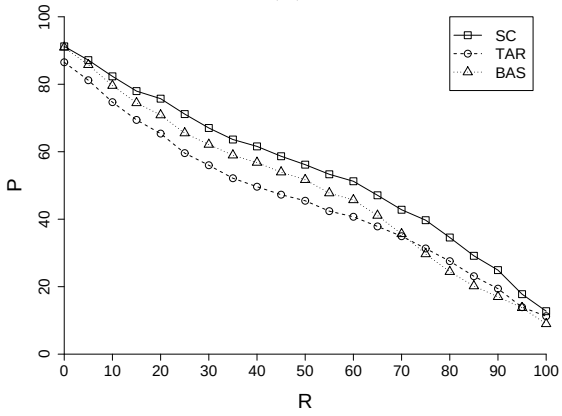
(a)

Bull-eye	MPEG7B		
Puntos	BAS	SC	TAR
64	76.58	**80.16**	79.04
100	77.43	**81.61**	79.25
128	77.74	**82.08**	79.31

(b)



(c)



(d)

Figura 8.24: Gráficos PR y test bull-eye para la comparativa entre los descriptores BAS, *shape contexts* y TAR, utilizando ATNLC como medida de disimilitud. (a) Gráfico PR para MPEG7B, (b) Bull-eye, (c) gráfico PR para Silhouette y (d) gráfico PR para SQUID. Todos los gráficos PR son de contornos de 128 puntos.

MAP	MPEG7B			Silhouette			SQUID		
Puntos	BAS	SC	TAR	BAS	SC	TAR	BAS	SC	TAR
64	70.81	**75.73**	72.83	73.46	**83.01**	70.85	46.75	**51.07**	44.66
100	71.88	**77.18**	73.11	74.53	**84.00**	71.15	47.61	**53.30**	44.68
128	72.26	**77.49**	73.17	74.88	**84.51**	71.16	48.05	**53.40**	44.76

Figura 8.25: MAP para la comparativa entre los descriptores BAS, *shape contexts* y TAR, utilizando ATNLC como medida de disimilitud.

8.3.7. Otras ponderaciones en el ATNLC

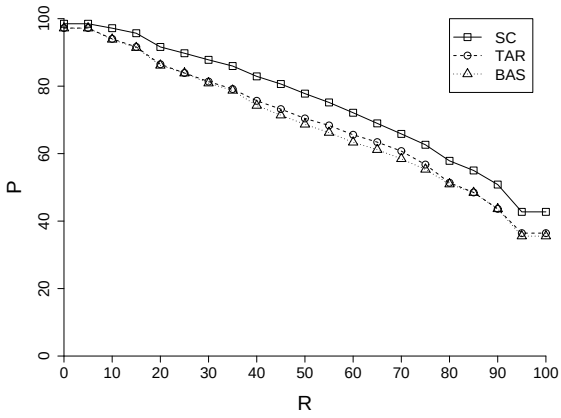
En los experimentos anteriores se ha utilizado la ponderación $\gamma = (1, 1, 1)$. En esta sección presentamos resultados de reconocimiento para las ponderaciones $\gamma = (1, 2, 1)$ y $\gamma = (1, 3, 1)$ (véase la Sección 5.3).

En las Figuras 8.26, 8.27 y 8.28 se muestran los resultados para la ponderación $\gamma = (1, 2, 1)$.

En las Figuras 8.29, 8.30 y 8.31 se muestran los resultados para la ponderación $\gamma = (1, 3, 1)$.

Clasif.	**MPEG7b**			**Silhouette**			**SQUID**		
Puntos	**BAS**	**SC**	**TAR**	**BAS**	**SC**	**TAR**	**BAS**	**SC**	**TAR**
64	94.57	**97.71**	96.21	95.42	**98.78**	94.67	74.09	74.90	**75.30**
100	95.36	**98.21**	96.21	95.88	**98.78**	95.04	78.14	**80.16**	73.68
128	95.93	**97.86**	96.21	96.26	**98.69**	94.76	78.54	**80.97**	75.30

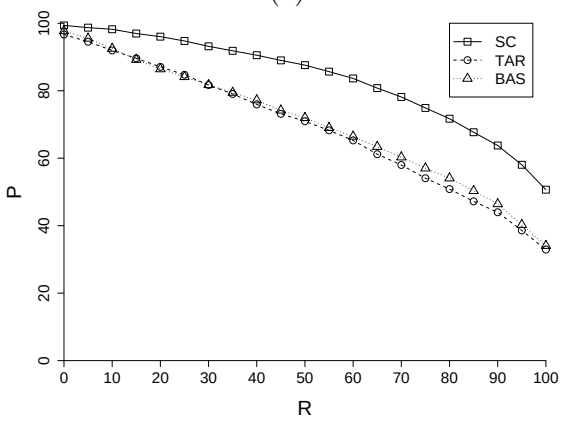
Figura 8.26: Tasas de clasificación (porcentaje de aciertos) para los descriptores BAS, *shape contexts* y TAR, seleccionando los puntos con el x-espaciado cíclico y utilizando ATNLC con la ponderación $\gamma = (1, 2, 1)$.



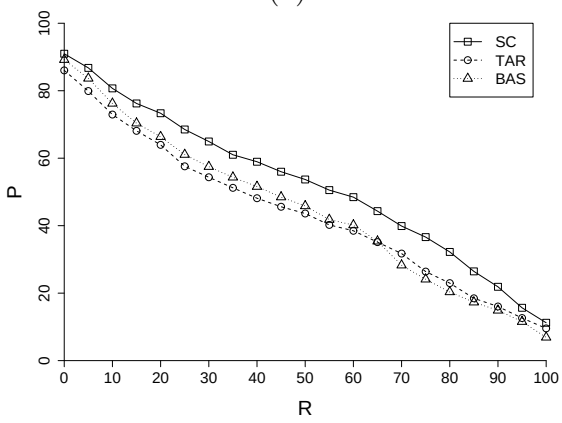
(a)

Bull-eye	MPEG7b		
Puntos	BAS	SC	TAR
64	72.95	**78.59**	75.19
100	74.40	**79.73**	75.38
128	74.78	**80.24**	75.50

(b)



(c)



(d)

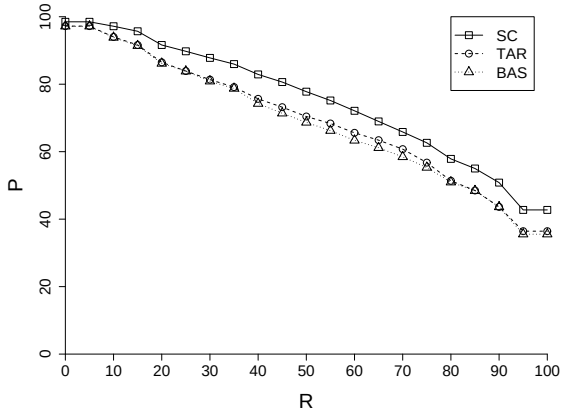
Figura 8.27: Gráficos PR y test bull-eye para los descriptores BAS, *shape contexts* y TAR, seleccionando los puntos con el x-espaciado cíclico y utilizando ATNLC con la ponderación $\gamma = (1, 2, 1)$. (a) Gráfico PR para MPEG7b, (b) Bull-eye, (c) gráfico PR para Silhouette y (d) gráfico PR para SQUID. Todos los gráficos PR son de contornos de 128 puntos.

MAP	MPEG7b			Silhouette			SQUID		
Puntos	BAS	SC	TAR	BAS	SC	TAR	BAS	SC	TAR
64	66.00	**73.62**	69.18	67.96	**82.20**	68.20	42.72	**48.95**	42.68
100	67.88	**75.09**	69.49	69.39	**83.00**	68.46	43.55	**50.91**	42.53
128	68.56	**75.48**	69.52	69.77	**83.46**	68.46	43.77	**51.16**	42.62

Figura 8.28: MAP para los descriptores BAS, *shape contexts* y TAR, seleccionando los puntos con el x-espaciado cíclico y utilizando ATNLC con la ponderación $\gamma = (1, 2, 1)$.

Clasif.	MPEG7b			Silhouette			SQUID		
Puntos	**BAS**	**SC**	**TAR**	**BAS**	**SC**	**TAR**	**BAS**	**SC**	**TAR**
64	90.71	**97.57**	95.64	93.17	**98.50**	94.48	72.47	75.30	**75.71**
100	94.00	**97.93**	96.14	94.95	**98.50**	94.67	77.33	**80.57**	74.49
128	94.93	**97.64**	96.21	95.70	**98.69**	94.76	77.73	**80.97**	74.49

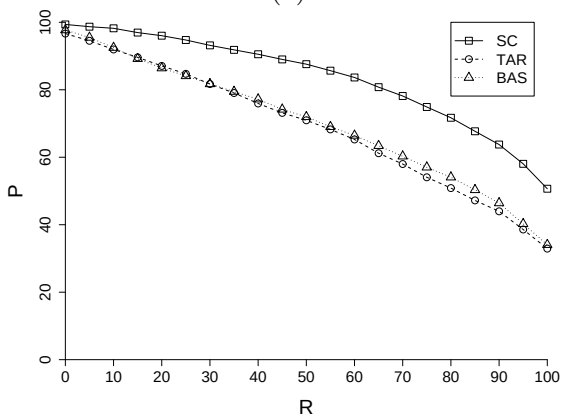
Figura 8.29: Tasas de clasificación (porcentaje de aciertos) para los descriptores BAS, *shape contexts* y TAR, seleccionando los puntos con el x-espaciado cíclico y utilizando ATNLC con la ponderación $\gamma = (1, 3, 1)$.



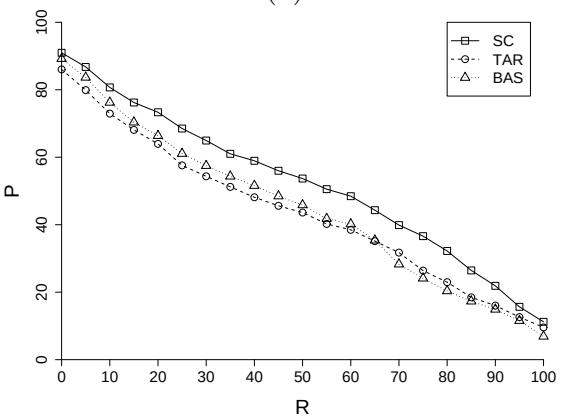
(a)

Bull-eye	MPEG7b		
Puntos	**BAS**	**SC**	**TAR**
64	68.18	**76.49**	73.45
100	71.80	**78.29**	74.54
128	73.11	**78.87**	74.85

(b)



(c)



(d)

Figura 8.30: Gráficos PR y test bull-eye para los descriptores BAS, *shape contexts* y TAR, seleccionando los puntos con el x-espaciado cíclico y utilizando ATNLC con la ponderación $\gamma = (1, 3, 1)$. (a) Gráfico PR para MPEG7b, (b) Bull-eye, (c) gráfico PR para Silhouette y (d) gráfico PR para SQUID. Todos los gráficos PR son de contornos de 128 puntos.

MAP	**MPEG7b**			**Silhouette**			**SQUID**		
Puntos	**BAS**	**SC**	**TAR**	**BAS**	**SC**	**TAR**	**BAS**	**SC**	**TAR**
64	60.58	**71.36**	67.60	62.92	**80.32**	66.82	41.77	**48.60**	42.24
100	65.08	**73.54**	68.69	66.64	**81.74**	67.69	43.24	**50.62**	42.29
128	66.57	**74.16**	68.97	67.73	**82.48**	67.92	43.52	**50.99**	42.44

Figura 8.31: MAP para los descriptores BAS, *shape contexts* y TAR, seleccionando los puntos con el x-espaciado cíclico y utilizando ATNLC con la ponderación $\gamma = (1, 3, 1)$.

8.4. Experimentos con las técnicas de aceleración

Presentamos aquí experimentos para la evaluación de las técnicas de aceleración. Todos los experimentos se han hecho sobre una máquina Intel i7 a 2.66GHz corriendo bajo linux 2.6.31. En principio, se muestran sólo resultados para 100 puntos, por ser los más utilizados en la literatura y, en general, la selección de 128 puntos no conlleva una mejora considerable, tal como se ha visto en la sección anterior.

En la sección anterior hemos visto también que la ciclicidad es necesaria y que los descriptores de puntos con información global ofrecen resultados muy competitivos. Sin embargo, ambas cosas, sobre todo la segunda, hacen que el método de comparación sea muy costoso. Por ejemplo, para 100 puntos, comparar una muestra contra las 1399 muestras restantes del corpus MPEG7 (parte B) (suponiendo que queramos clasificar o recuperar formas para esa muestra) con el ATNLC, tiene un tiempo (medio) de 2 segundos para la curvatura, 3.5 segundos para el BAS, 6 segundos para el TAR y 9 segundos para los *shape contexts*, aproximadamente. En esta sección veremos cómo mejoran estos tiempos con las técnicas del Capítulo 6.

En primer lugar se verá experimentalmente cómo la dimensionalidad intrínseca afecta al cumplimiento de la desigualdad triangular. Posteriormente mostraremos los resultados de aceleración con BBATNLC, AESA, LAESA y LAESAEA. Finalmente veremos cómo nos ha afectado el uso de los métodos basados en AESA en las tasas de acierto debido al incumplimiento por parte del ATNL de la desigualdad triangular.

8.4.1. Dimensionalidad intrínseca y desigualdad triangular

Para ver que efectivamente la dimensionalidad intrínseca y el cumplimiento de la desigualdad triangular están relacionados (véase la Sección 6.3.3) mostramos primero unas tablas en las Figuras 8.32, 8.33 y 8.34, una por cada corpus. En ellas se muestra la dimensionalidad intrínseca y el correspondiente porcentaje de tripletas que no cumplen la desigualdad para cada uno de los descriptores de contorno.

Como puede observarse a medida que sube la dimensionalidad intrínseca baja el porcentaje de tripletas que violan la desigualdad triangular. Pero también vemos que pueden intervenir otros factores, como el tipo de descriptor, en el caso del TAR. Con los corpus Silhouette y SQUID, el descriptor TAR cumple siempre la desigualdad triangular, cosa que no pasa con los *shape contexts*, a pesar de que estos últimos tienen una dimensionalidad intrínseca más alta. Como se comenta en la Sección 6.3.3 la dimensionalidad intrínseca sólo hace que sea menos probable el incumplimiento de la desigualdad triangular.

En las Figuras 8.35, 8.36, 8.37, 8.38, 8.39 y 8.40 podemos ver mediante histogramas la relación de la distribución de distancias con la distribución de H (véase la Sección 6.3.3).

MPEG7B	ϱ	**violaciones**
Curvatura	0.025	2.95 %
BAS	2.522	$7.25 \cdot 10^{-3}$ %
TAR	6.231	$3.83 \cdot 10^{-4}$ %
SC	17.596	$7.07 \cdot 10^{-5}$ %

Figura 8.32: Comparación de la dimensionalidad intrínseca con el porcentaje de tripletas que violan la desigualdad triangular para el corpus MPEG7B.

Silhouette	ϱ	**violaciones**
Curvatura	0.652	$3.93 \cdot 10^{-1}$ %
BAS	8.812	$2.61 \cdot 10^{-5}$ %
TAR	14.072	0.00 %
SC	19.545	$6.54 \cdot 10^{-7}$ %

Figura 8.33: Comparación de la dimensionalidad intrínseca con el porcentaje de tripletas que violan la desigualdad triangular para el corpus Silhouette.

SQUID	ϱ	**violaciones**
Curvatura	0.023	$4.19 \cdot 10^{-1}$ %
BAS	3.489	$1.33 \cdot 10^{-4}$ %
TAR	5.878	0.00 %
SC	12.022	$1.50 \cdot 10^{-7}$ %

Figura 8.34: Comparación de la dimensionalidad intrínseca con el porcentaje de tripletas que violan la desigualdad triangular para el corpus SQUID.

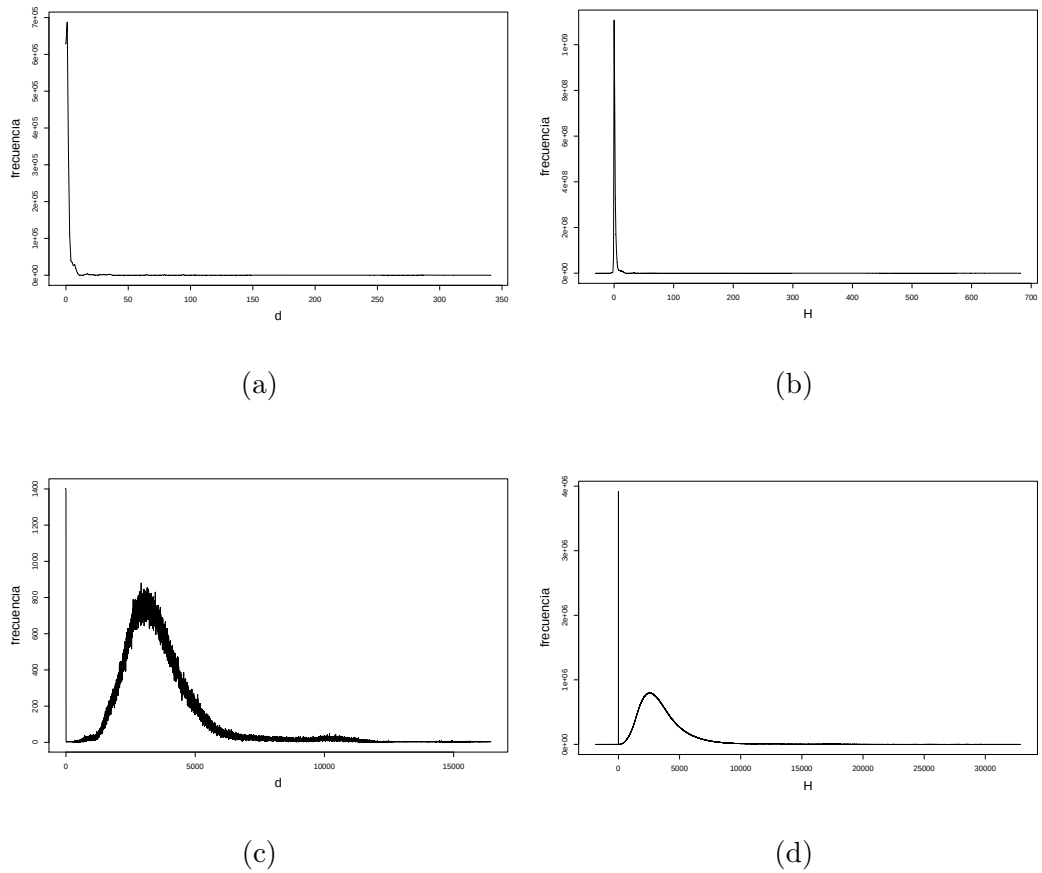


Figura 8.35: Histogramas de la distribución de distancias (izquierda) y la distribución de H (derecha), para el corpus MPEG7B, con (a, b) la curvatura y (c, d) BAS.

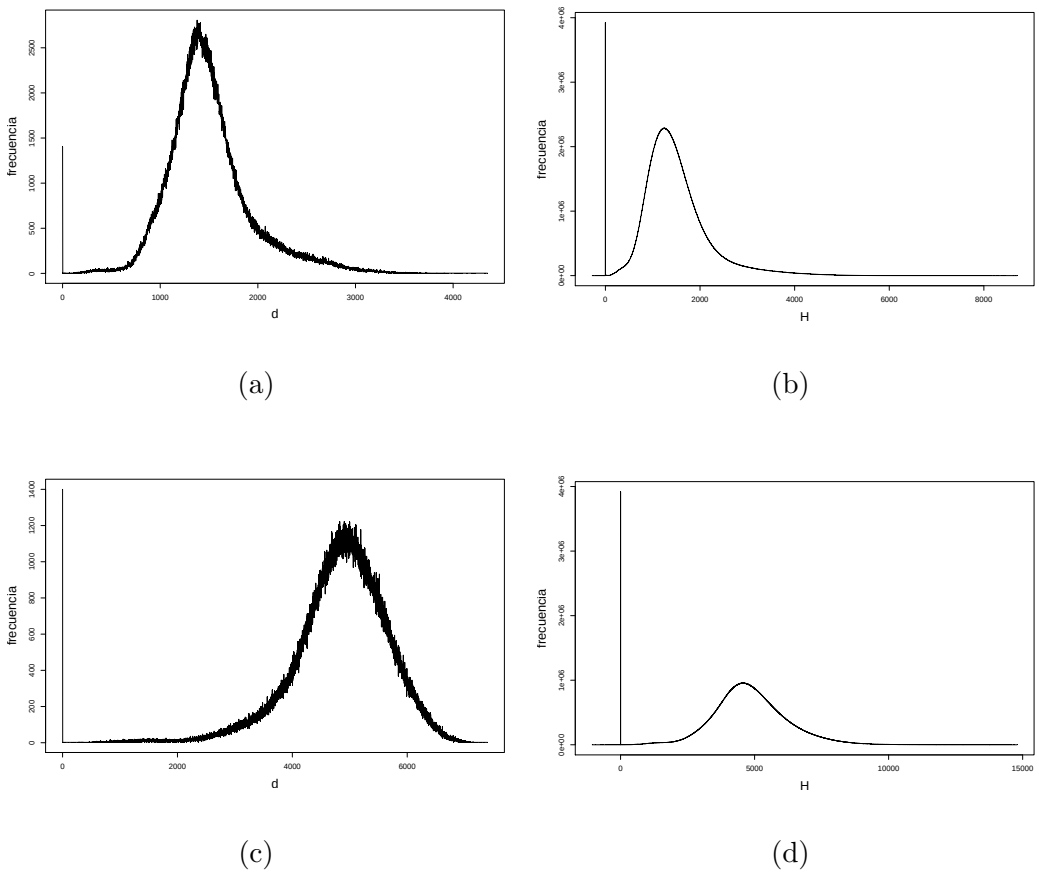


Figura 8.36: Histogramas de la distribución de distancias (izquierda) y la distribución de H (derecha), para el corpus MPEG7B, con (a, b) TAR y (c, d) los *shape contexts*.

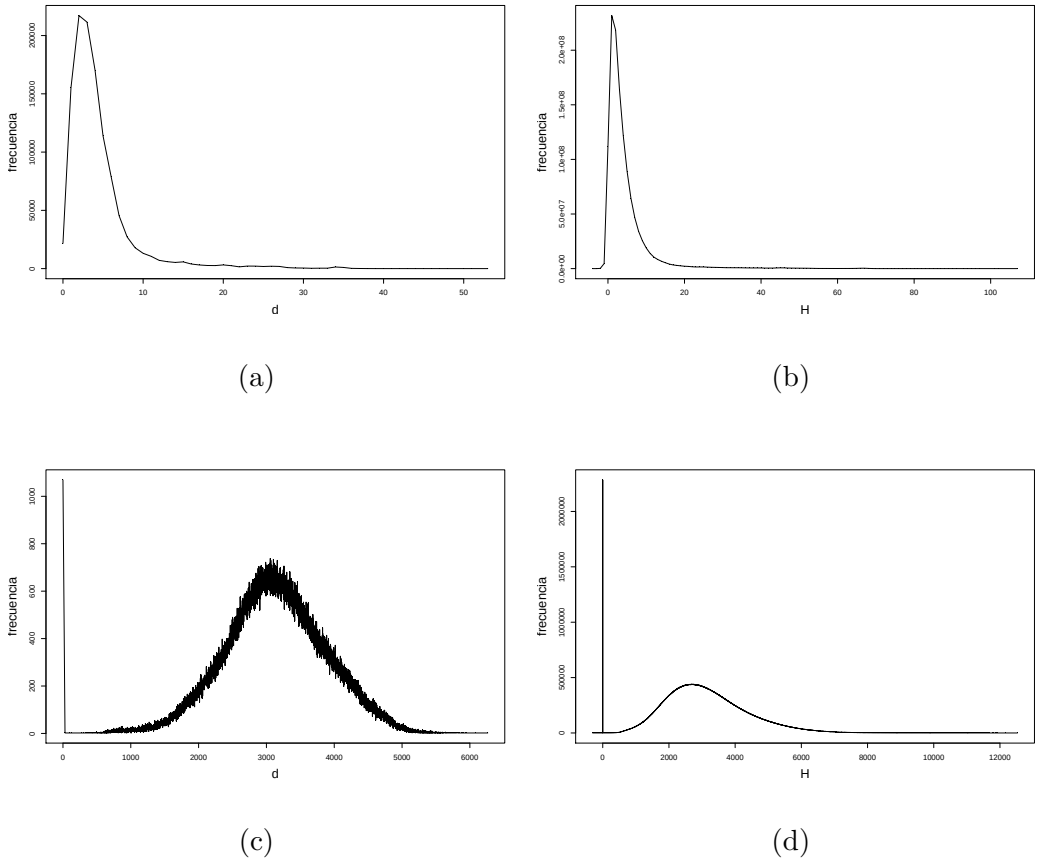


Figura 8.37: Histogramas de la distribución de distancias (izquierda) y la distribución de H (derecha), para el corpus Silhouette, con (a, b) la curvatura y (c, d) BAS.

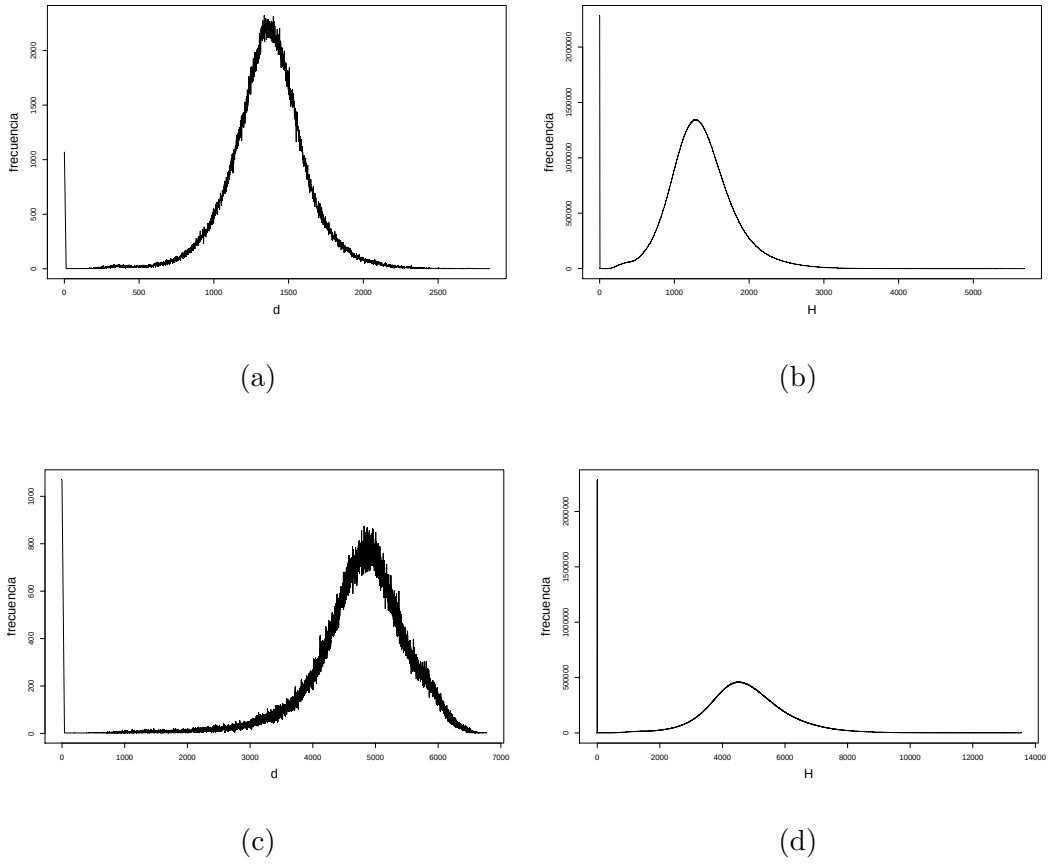


Figura 8.38: Histogramas de la distribución de distancias (izquierda) y la distribución de H (derecha), para el corpus Silhouette, con (a, b) TAR y (c, d) los *shape contexts*.

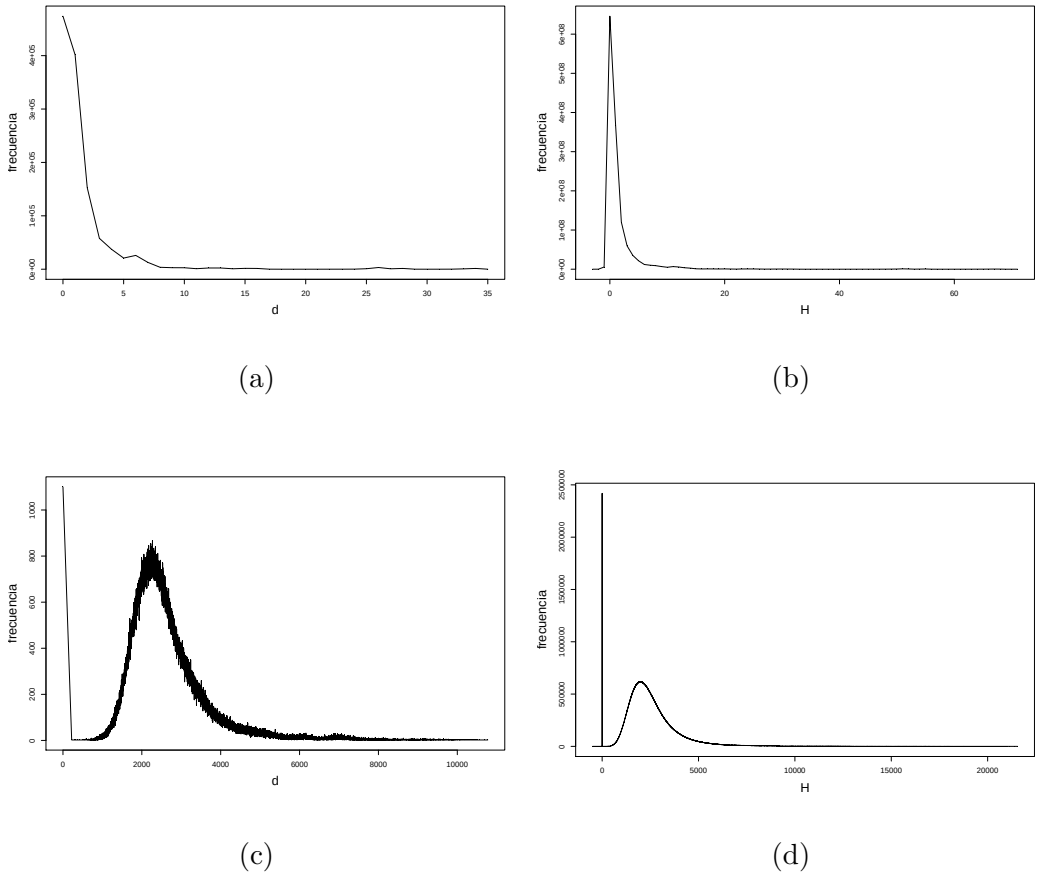


Figura 8.39: Histogramas de la distribución de distancias (izquierda) y la distribución de H (derecha), para el corpus SQUID, con (a, b) la curvatura y (c, d) BAS.

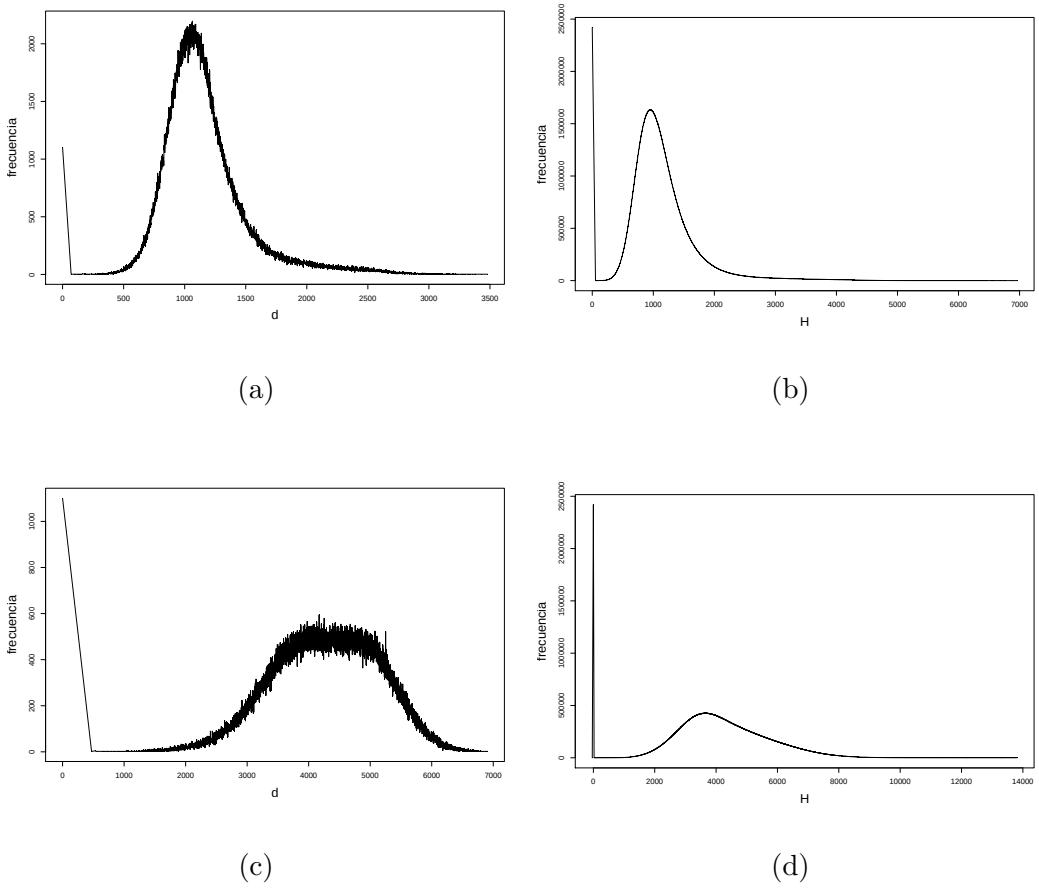


Figura 8.40: Histogramas de la distribución de distancias (izquierda) y la distribución de H (derecha), para el corpus SQUID, con (a, b) TAR y (c, d) los *shape contexts*.

8.4.2. Tiempos

Hemos hecho experimentos de recuperación de las k formas más similares, con estos valores de k: 1, 5, 10, 20 y 40. En el caso de la clasificación muchas veces sería bastante con $k = 1$, aunque también podrían utilizarse valores mayores, según el caso. Una recuperación de 10 o 20 prototipos podría ser suficiente como primera (o incluso única) respuesta a un usuario de una aplicación concreta de recuperación de formas.

Para cada corpus y cada descriptor presentamos dos gráficos con los resultados para BBATNLC, k-AESA, k-LAESA y k-LAESAEA con respecto a la búsqueda exhaustiva. En el primero tenemos los tiempos totales y en el segundo las distancias ATNLC calculadas, es decir, cuántas veces se ha tenido que recurrir a el cálculo de una distancia ATNLC. Puede verse así (en el caso de BBATNLC y LAESAEA) que a medida que el descriptor es más pesado cobra menos importancia el procedimiento divide y vencerás del ATNLC con respecto al cálculo de distancias locales.

El número de prototipos base seleccionados (con el procedimiento que aparece en [MOV94], pero realizando la búsqueda a intervalos de 10 en 10) para los métodos k-LAESA y k-LAESAEA se muestra en las Figuras 8.41, 8.42 y 8.43.

Los gráficos para el corpus MPEG7B están en las Figuras 8.44 y 8.45. Los del corpus Silhouette en las Figuras 8.46 y 8.47. Y los del corpus SQUID en las Figuras 8.48 y 8.49.

Nótese que el ahorro del BBATNLC (el método óptimo) oscila entre el 50 % y el 35 % en todos los casos, para $k = 1$. Lo cual lo hace una opción interesante para la clasificación. También parece interesante en la recuperación de formas para los casos de la curvatura y el BAS, ya que el ahorro no baja a menos del 25 % para $k = 40$.

Con los métodos basados en AESA se suele obtener muy buenos resultados tanto para clasificación como para recuperación, destacando el AESA, tal como ocurre en la literatura [MOV94, JM03]. LAESAEA supera claramente al LAESA y nos puede ser útil cuando necesitemos disminuir la complejidad espacial. En el caso de la curvatura, AESA, LAESA y LAESAEA obtienen prácticamente el mismo resultado, pero no parecen ser una buena opción para este tipo de descriptor tal como veremos en la Sección 8.4.3.

k	1	5	10	20	40
Curvatura	10	10	10	10	10
BAS	20	60	60	70	70
TAR	40	100	100	100	100
SC	70	150	150	150	150

Figura 8.41: Número de prototipos base seleccionados para los métodos k-LAESA y k-LAESAEA, para cada valor de k y cada descriptor, curvatura, BAS, TAR y los *shape contexts* (SC), con el corpus MPEG7B.

k	1	5	10	20	40
Curvatura	30	30	30	40	40
BAS	60	60	60	90	90
TAR	60	60	60	90	90
SC	80	150	150	150	150

Figura 8.42: Número de prototipos base seleccionados para los métodos k-LAESA y k-LAESAEA, para cada valor de k y cada descriptor, curvatura, BAS, TAR y los *shape contexts* (SC), con el corpus Silhouette.

k	1	5	10	20	40
Curvatura	20	30	30	30	30
BAS	60	60	60	60	60
TAR	50	50	50	50	50
SC	180	200	200	200	200

Figura 8.43: Número de prototipos base seleccionados para los métodos k-LAESA y k-LAESAEA, para cada valor de k y cada descriptor, curvatura, BAS, TAR y los *shape contexts* (SC), con el corpus SQUID.

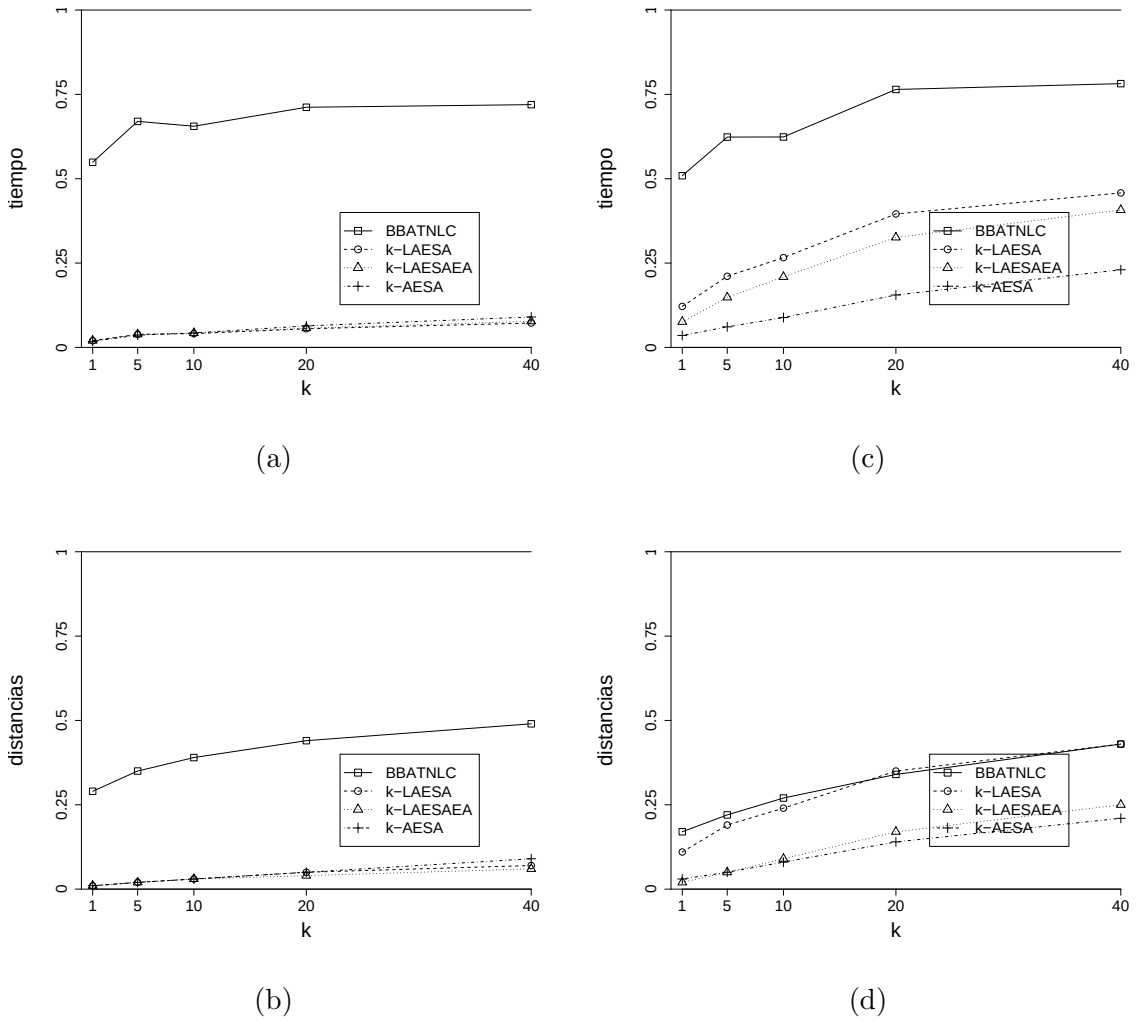


Figura 8.44: Tiempos relativos y número relativo de distancias ATNLC calculadas con el corpus MPEG7B para (a) y (b) la curvatura y, (c) y (d) BAS. El valor 1 corresponde a la búsqueda exhaustiva.

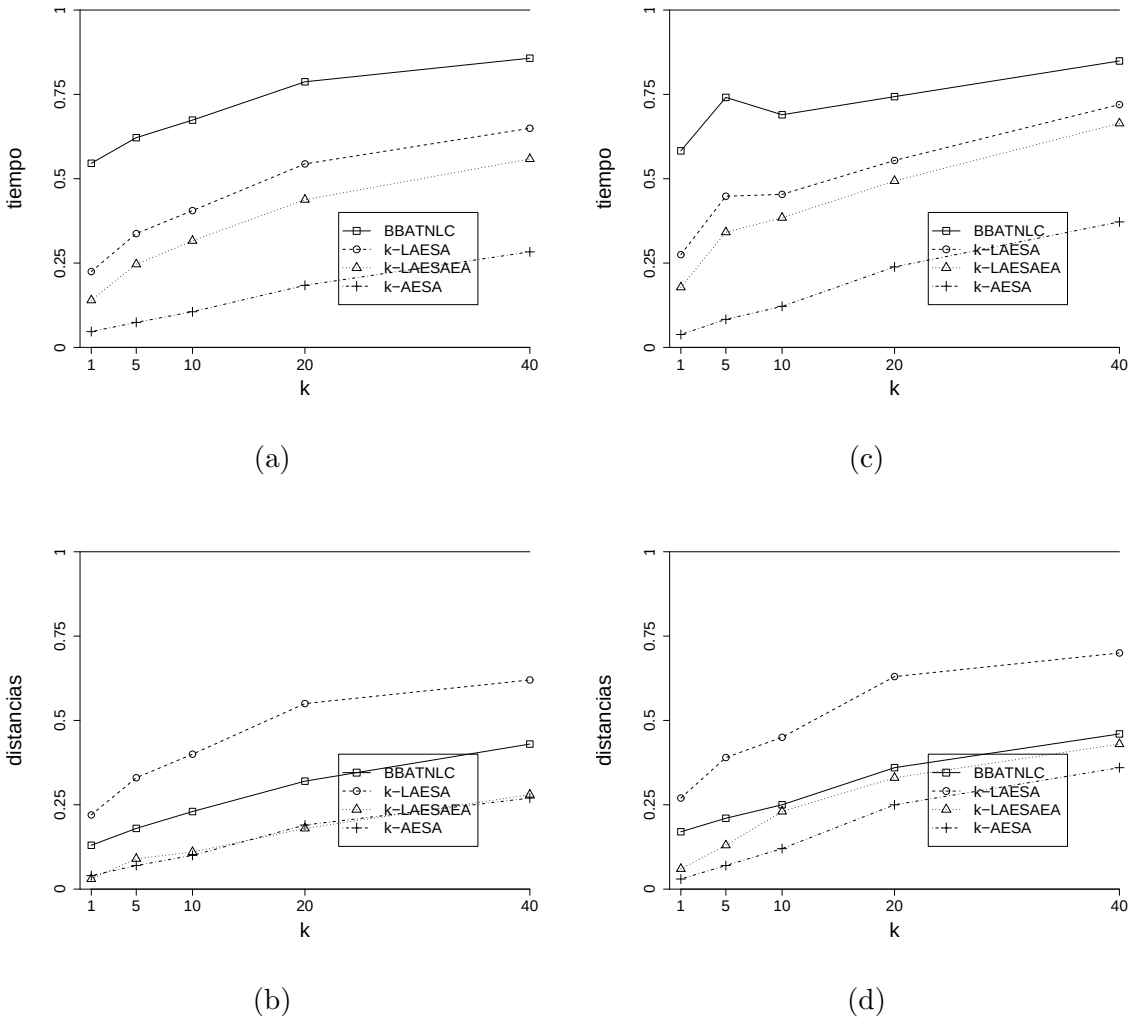


Figura 8.45: Tiempos relativos y número relativo de distancias ATNLC calculadas con el corpus MPEG7B para (a) y (b) TAR y, (c) y (d) los *shape contexts*. El valor 1 corresponde a la búsqueda exhaustiva.

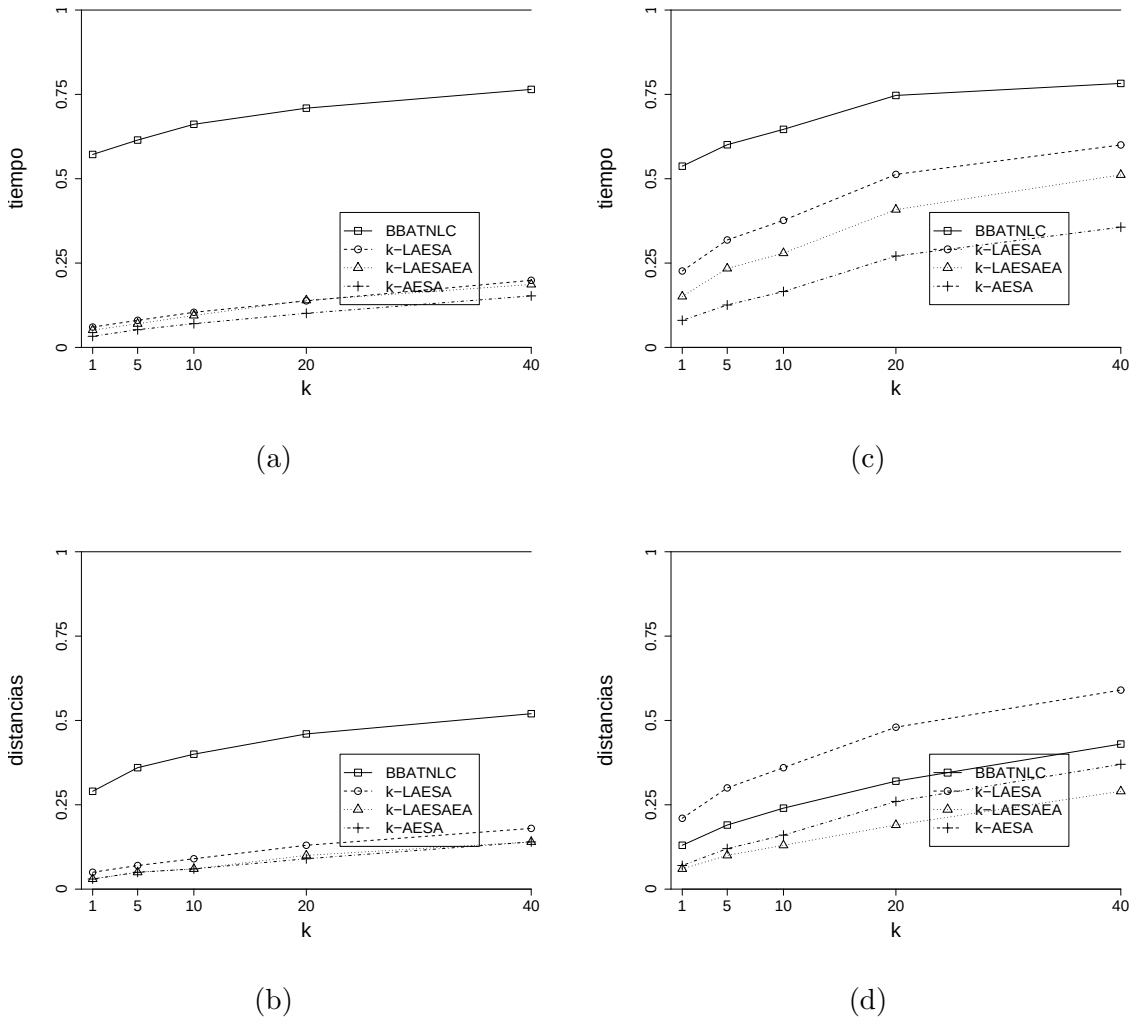


Figura 8.46: Tiempos relativos y número relativo de distancias ATNLC calculadas con el corpus Silhouette para (a) y (b) la curvatura y, (c) y (d) BAS. El valor 1 corresponde a la búsqueda exhaustiva.

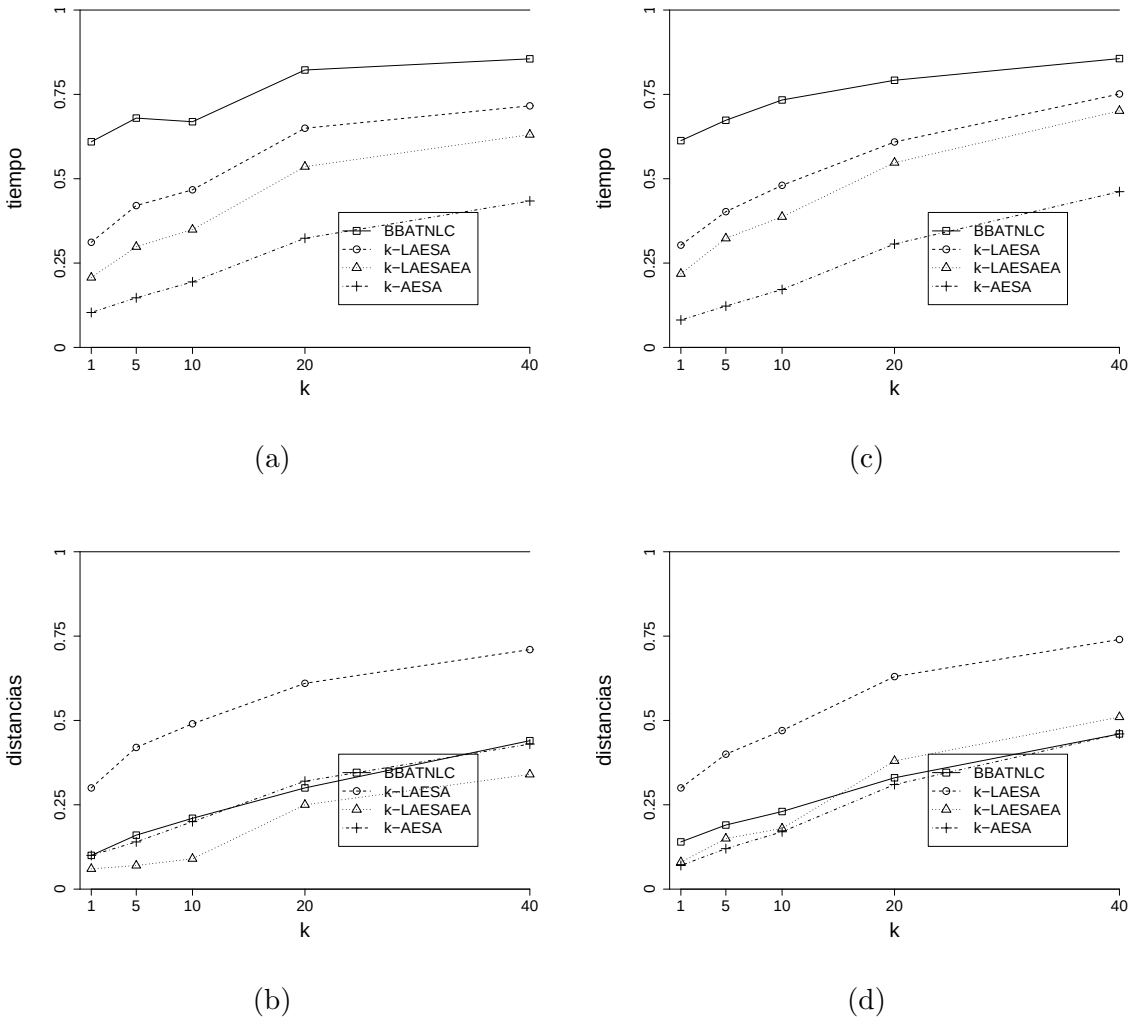


Figura 8.47: Tiempos relativos y número relativo de distancias ATNLC calculadas con el corpus Silhouette para (a) y (b) TAR y, (c) y (d) los *shape contexts*. El valor 1 corresponde a la búsqueda exhaustiva.

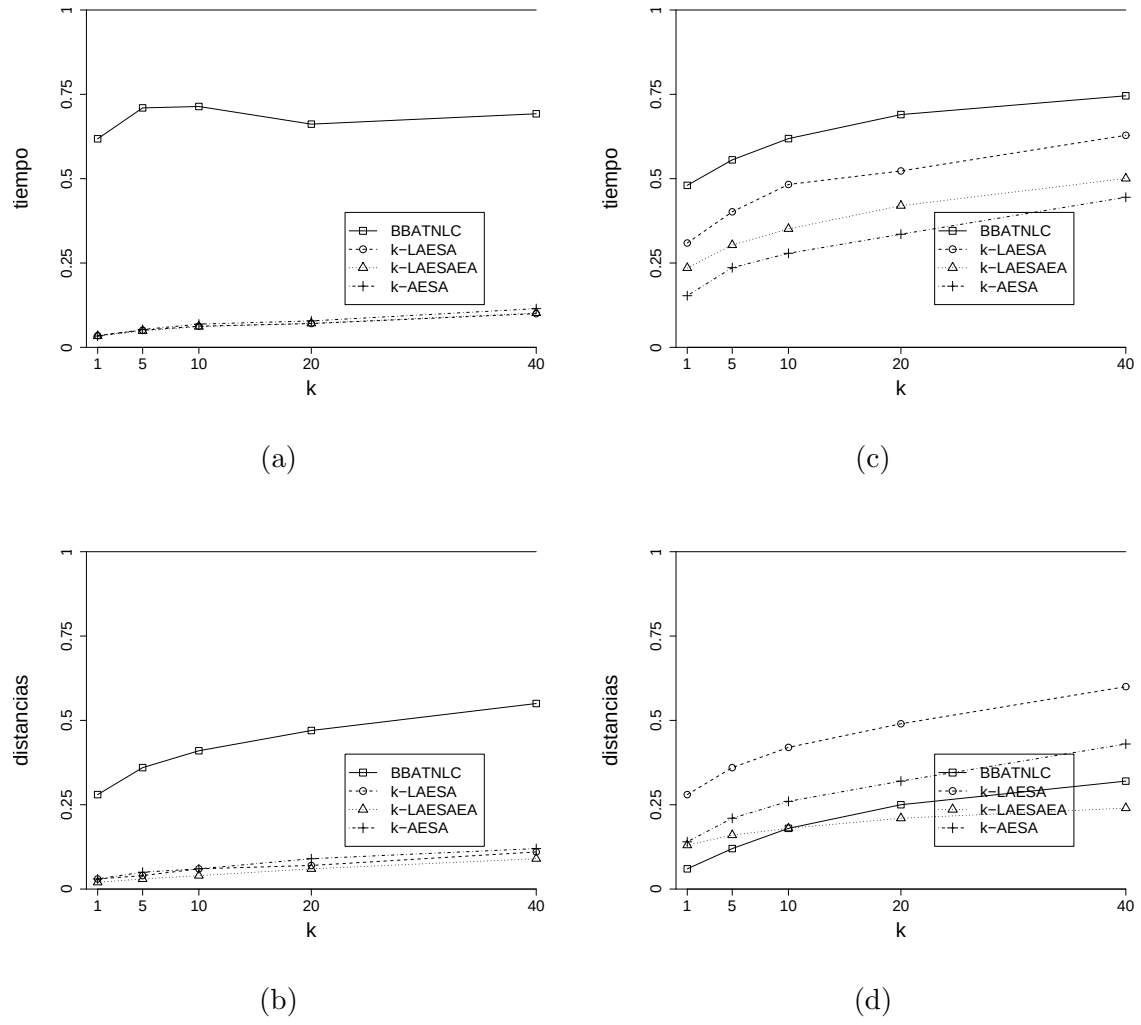


Figura 8.48: Tiempos relativos y número relativo de distancias ATNLC calculadas con el corpus SQUID para (a) y (b) la curvatura y, (c) y (d) BAS. El valor 1 corresponde a la búsqueda exhaustiva.

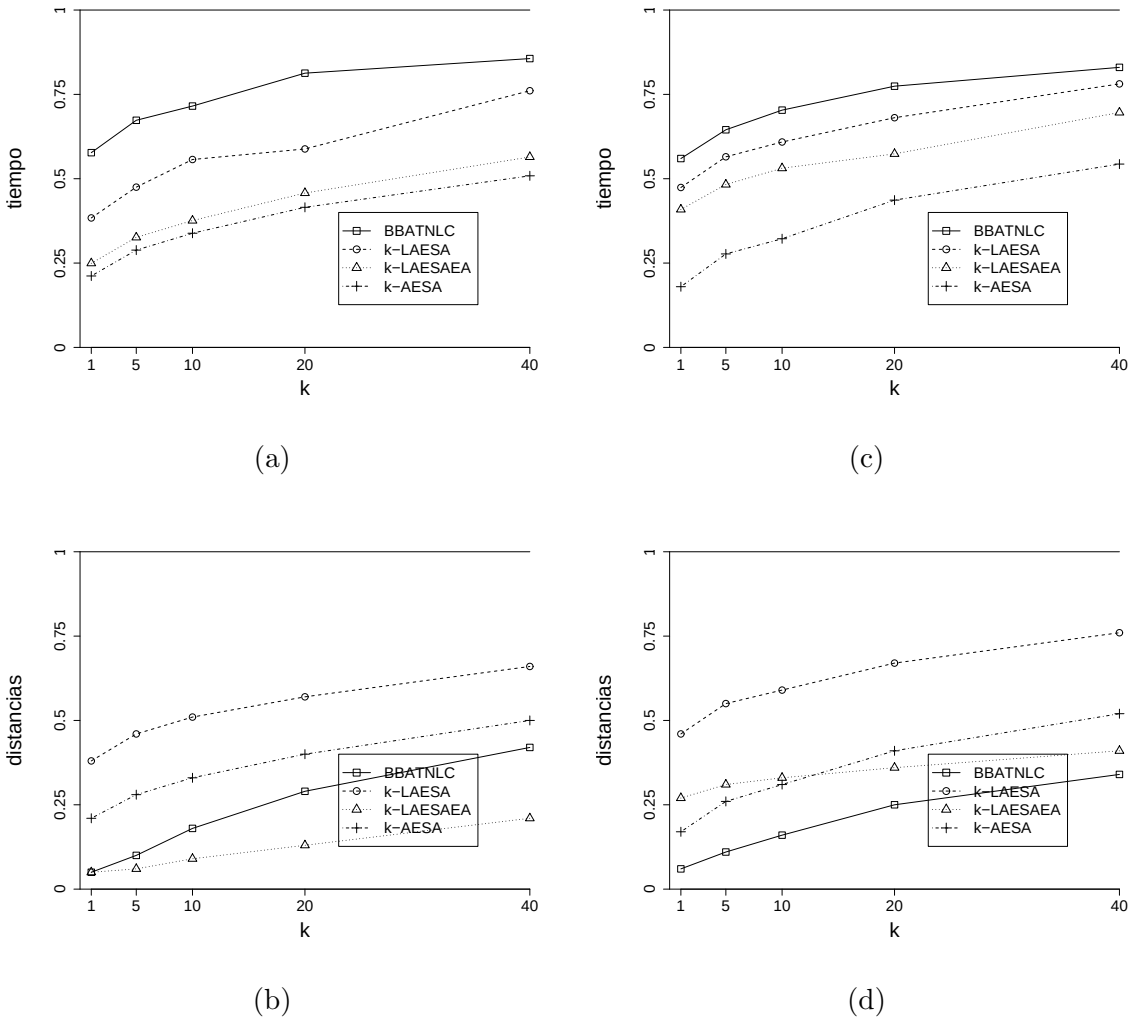


Figura 8.49: Tiempos relativos y número relativo de distancias ATNLC calculadas con el corpus SQUID para (a) y (b) TAR y, (c) y (d) los *shape contexts*. El valor 1 corresponde a la búsqueda exhaustiva.

8.4.3. Error de los métodos AESA con ATNLC

Para medir el error de los métodos basados en AESA con ATNLC, hemos hecho experimentos de recuperación de k formas, con valores de k: 1, 5, 10, 20 y 40. Se ha medido el porcentaje de aciertos que hay en los k vecinos más próximos. Es un test similar al *bull-eye test* [Bob01] pero con valores de k menores que 40 y además la k puede ser menor que la cantidad de prototipos que hay en una categoría.

Fijándonos sólo en los dos primeros decimales, los resultados con el descriptor TAR y los *shape contexts* no han cambiado. En el caso del BAS sólo ha habido un cambio en los dos decimales del corpus MPEG7B para el valor de $k = 40$. Los resultados con la curvatura se muestran en la Figura 8.50.

Vemos que los resultados de la curvatura no son malos a excepción de LAESA en el caso del corpus MPEG7B. En este corpus el número de tripletas que no cumplen la desigualdad triangular es de casi un 3 % y posiblemente la selección de prototipos base se produce sobre algunas muestras problemáticas. Con AESA estos prototipos han podido ser eliminados antes de aplicar la cota, de ahí que sus resultados no varíen tanto con respecto a la búsqueda exhaustiva.

	MPEG7B			**Silhouette**			**SQUID**		
k	**Exh.**	**AESA**	**LAESA**	**Exh.**	**AESA**	**LAESA**	**Exh.**	**AESA**	**LAESA**
1	90.50	90.00	85.42	91.77	91.87	90.40	84.81	84.81	85.00
5	83.21	83.23	78.11	88.25	88.25	88.35	85.91	85.95	85.52
10	70.94	70.74	64.25	80.27	80.27	80.06	81.38	81.38	81.18
20	55.39	55.29	49.52	69.47	69.47	68.89	77.48	77.44	77.46
40	63.83	63.43	56.24	61.20	61.20	60.78	75.92	75.88	75.68

Figura 8.50: Pérdida de porcentaje de aciertos del k-AESA y k-LAESA con respecto a la búsqueda exhaustiva, con la curvatura como descriptor de contorno.

8.5. Experimentos con los modelos ocultos de Markov

Presentamos aquí experimentos de clasificación con los modelos de Markov. Veremos que las extensiones a los modelos ocultos de Markov para tratar cadenas cíclicas son necesarias si nuestro objetivo es conseguir la invarianza al punto de inicio. Veremos también que la topología lineal es suficiente en este tipo de aplicaciones (reconocimiento de contornos), es decir, no necesitamos recurrir a otro tipo de topologías izquierda-derecha. Que esto ocurra hará posible utilizar los MOMLC (véase la Sección 7.5) con todas las ventajas que ello conlleva. Todo esto[4] se ha realizado utilizando la curvatura (véase la Sección 2.2.2.4).

La evaluación de las técnicas se ha hecho con tasas de clasificación, para diferente número de estados: 10, 30, 50, 70, 90, 110 y 128 estados. Hemos acabado en 128 estados debido a que utilizamos los contornos de 128 puntos. Por otro lado, sólo utilizamos una gaussiana por estado.

Todos los experimentos se han realizado con validación cruzada [DH73] y para la clasificación no hemos hecho uso de la probabilidad sino de la puntuación de Viterbi, tanto en el caso cíclico, como en el no cíclico.

8.5.1. Invarianza al punto inicial

En la Sección 7.1 se comentan varias soluciones de la literatura para tratar el problema de la invarianza al punto inicial. En esta sección comparamos estas soluciones con el heurístico APR propuesto en la Sección 7.4. En particular, nos comparamos con la topología circular de Arica et al. [AYV00], la elección del punto de inicio con los descriptores de Fourier [HK91] y la topología ergódica [BM04, TGJ07]. En el caso de nuestro heurístico tenemos dos versiones: en una utilizamos el heurístico otra vez (además de en el entrenamiento) para clasificar con un Viterbi convencional (no cíclico); y en la otra, utilizamos el Viterbi cíclico para este propósito.

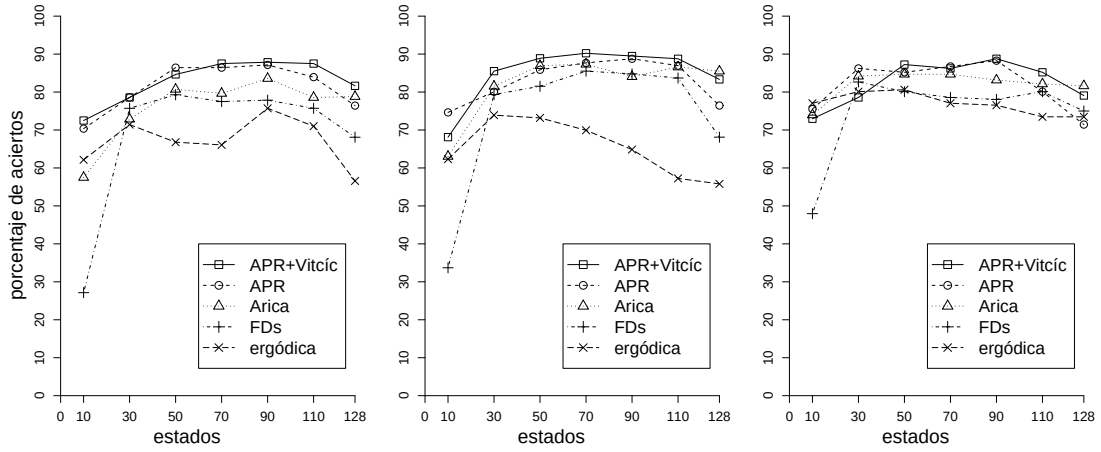
En las Figuras 8.51a, 8.51b y 8.51c se encuentran los resultados de la comparación, una por cada corpus. En ellos vemos que el entrenamiento con el heurístico APR y la clasificación con el Viterbi cíclico obtiene los mejores resultados. La topología ergódica, como era de esperar, obtiene los peores resultados[5]. La elección del punto inicial y la topología circular (en especial esta última) resultan ser los más competitivos frente al heurístico.

8.5.2. Otras topologías izquierda-derecha

En la Sección 7.1 mencionamos que las topologías izquierda-derecha son las más adecuadas para el modelado de cadenas. En la Sección 7.5 concretamos que, dentro de estas topologías, la lineal parece la mejor, ya que añadir más transiciones aumenta la complejidad del modelo. Aquí demostramos empíricamente esta afirmación con una comparativa entre tres topologías izquierda-derecha: la lineal, la de Bakis y la que tiene cuatro transiciones por estado. Esta última es la de Bakis más una transición al siguiente del siguiente

[4]Tal como hacen los métodos de reconocimiento de formas bidimensionales basados en MOMs de la literatura.

[5]Veremos más sobre esta topología en la Sección 8.5.4.



(a) (b) (c)

Figura 8.51: Tasas de clasificación para la comparativa entre la topología circular (Arica), la elección del punto inicial con los descriptores de Fourier (FDs), la topología ergódica, el heurístico (APR) y el heurístico con la clasificación del Viterbi cíclico (APR+Vitcíc). Con los corpus (a) MPEG7B, (b) Silhouette y (c) SQUID.

del siguiente estado. La técnica que hemos utilizado para entrenar y clasificar, en este caso, es la que mejor nos ha funcionado en la sección anterior: el entrenamiento con el heurístico y la clasificación con el Viterbi cíclico.

Los resultados se muestran en las Figuras 8.52a, 8.52b y 8.52c, una figura por cada corpus. Como puede verse la topología lineal supera a las otras.

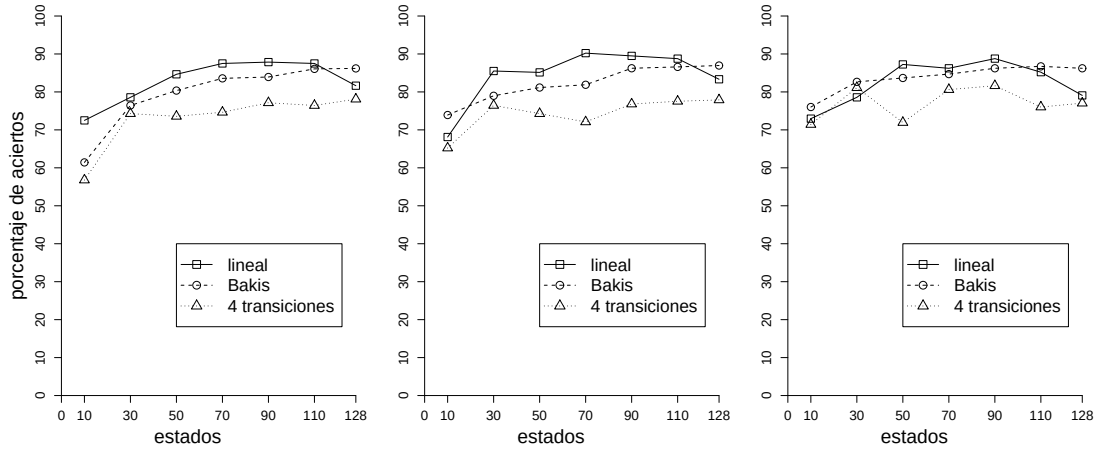
Para los experimentos de las secciones siguientes haremos solamente uso de esta topología lineal.

8.5.3. Entrenamiento cíclico

En esta sección comparamos el entrenamiento cíclico (Baum-Welch y Viterbi, véase la Sección 7.3) con el entrenamiento utilizando el heurístico APR. También tenemos en cuenta el entrenamiento con una buena inicialización (véase la Sección 7.6) o sin esta. En todos los casos la clasificación se realiza con el Viterbi cíclico.

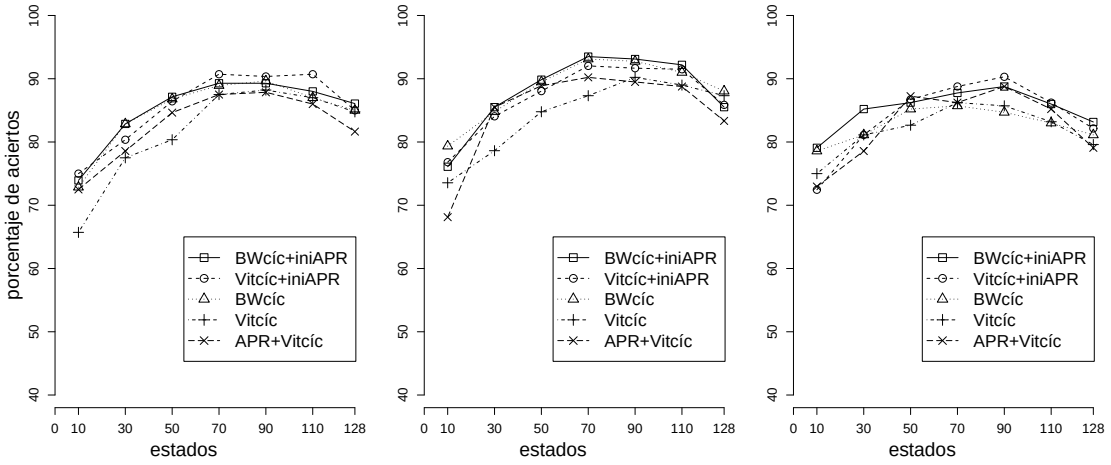
Los resultados de la comparación están en las Figuras 8.53a, 8.53b y 8.53c. Una para cada corpus. Como se puede observar suelen predominar los casos en los que, o gana el Baum-Welch cíclico (con inicialización), o el Viterbi cíclico (con inicialización), sin alejarse demasiado el uno del otro. Por otro lado, el entrenamiento cíclico con Viterbi parece más sensible a una buena inicialización, el Baum-Welch en la mayoría de casos apenas mejora.

Cabe destacar aquí que, aunque ambas técnicas obtienen resultados parecidos, el entrenamiento con Baum-Welch cíclico es más costoso ya que (teniendo en cuenta que tenemos topologías lineales) si utilizamos el Viterbi cíclico para entrenar podemos hacer uso de las técnicas de la Sección 7.5 y así, conseguir un coste de $O(Lmn \log n)$ para cada iteración.



(a) (b) (c)

Figura 8.52: Tasas de clasificación para la comparativa entre diversas topologías izquierda-derecha. Topología lineal, topología de Bakis y topología de 4 transiciones. Para entrenar utilizamos el heurístico y para clasificar el Viterbi cíclico. Con los corpus (a) MPEG7B, (b) Silhouette y (c) SQUID.



(a) (b) (c)

Figura 8.53: Tasas de clasificación para la comparativa entre los entrenamientos: Baum-Welch cíclico con inicialización (BWcíc+iniAPR), Viterbi cíclico con inicialización (Vitcíc+iniAPR), Baum-Welch cíclico (BWcíc), Viterbi cíclico (Vitcíc) y el heurístico (APR+Vitcíc). Para clasificar se ha utilizado el Viterbi cíclico. Con los corpus (a) MPEG7B, (b) Silhouette y (c) SQUID.

8.5.4. Más sobre la topología ergódica

En la Sección 8.5.1 hemos visto experimentalmente que la topología ergódica no ofrece buenos resultados. Sin embargo, en la literatura existen trabajos [BM04, BMF04, TGJ07] en los que se utiliza esta topología alegando y demostrando sus buenas cualidades en la clasificación de contornos.

Más concretamente, en [BM04] se hacen experimentos con esta topología. Para entrenar, se realiza una selección del número de estados con BIC (*Bayesian Inference Criterion*) [Sch78] sobre un agrupamiento de curvaturas. Se obtienen buenos resultados pero las bases de datos son pequeñas y simples, con pocas muestras y clases. Utiliza un subconjunto del corpus MPEG7B de 6 clases con 10 muestras por clase[6] (llamemos a esto corpus L_2, tal como se hace en [TGJ07]). Utiliza también un corpus que aparece en [HK91] (de 8 clases con 1 muestra cada una) para hacer un experimento de invarianza al punto de inicio obteniendo un 100 %. Así, concluye que los modelos de Markov con la topología ergódica son suficientes para obtener esta invarianza. En este experimento mediante el entrenamiento con el Viterbi cíclico (con inicialización) nosotros también obtenemos un 100 %.

En [BMF04], un trabajo de los mismos autores, se hace uso también del mismo subconjunto L_2. En este caso, sí que se utiliza un heurístico para la elección de un punto de inicio, contradiciendo el anterior trabajo. Tampoco se usa el BIC para obtener el número de estados sino que se utiliza un número fijo de estados.

En [TGJ07], se parte de los trabajos anteriores [BM04, BMF04] pretendiendo mejorarlos cambiando el entrenamiento por uno basado en el método GPD (*Generalized probabilistic descent method*). Este autor vuelve a utilizar el corpus L_2 y crea uno nuevo, el L_1, siendo este las 6 clases pero con todas las muestras, es decir, un total de 20. Con el corpus L_2 consigue un 97.63 % (en [BMF04] se obtiene un 98.8 %). Con el corpus L_1 consigue un 96.43 %. Nosotros con el corpus L_1 y el entrenamiento cíclico con Viterbi (con inicialización) hemos llegado a un 99.28 %, superando incluso lo obtenido en [BMF04] para L_2.

Ninguno de los anteriores trabajos muestra resultados con la totalidad del corpus MPEG7B.

[6] El corpus MPEG7, tal como se dice en la Sección 8.2.1.1, contiene 70 clases con 20 muestras cada una.

Parte III

Discusión final

Capítulo 9

Discusión final y desarrollos futuros

Porco Rosso: «*I'd rather be a pig than a fascist.*»
Hayao Miyazaki, *Pòrco Rósso* (1998).

Concluimos ya con las aportaciones de la tesis y las líneas futuras de investigación. Comentaremos aquí también las publicaciones relacionadas con la tesis.

9.1. Aportaciones de la tesis

El problema central que hemos abordado en esta tesis fue mencionado en la Sección 1.5 de la siguiente manera. Cuando queremos comparar dos formas bidimensionales se nos presenta un problema que entraña cierta dificultad: la invarianza al punto inicial. La única manera de conseguir esta invarianza es utilizando cadenas cíclicas.

Vistos los desarrollos teóricos expuestos en los Capítulos 5, 6 y 7, y los resultados experimentales del Capítulo 8, podemos considerar que hemos aportados soluciones muy interesantes para mejorar el tratamiento de estas cadenas cíclicas en el ámbito del ATNL y de los modelos ocultos de Markov.

Más en concreto, para lograr nuestro objetivo, con el ATNL:

- Hemos formalizado el ATNLC y corregido un error común de la literatura acerca de su cálculo, desarrollando además un algoritmo divide y vencerás eficiente para su cómputo (véase el Capítulo 5). Experimentalmente (Sección 8.3) hemos visto que la invarianza al punto de inicio sólo es posible con un tratamiento cíclico y que el ATNL (junto con los descriptores con puntos con información global) resulta muy competitivo frente a otras técnicas de reconocimiento. Por tanto, el uso del ATNLC queda justificado.
- Hemos planteado varios esquemas para acelerar el ATNLC, tanto en clasificación, como en recuperación de formas:
  - Un heurístico (APR, véase la Sección 6.1), sólo en el caso de que hayan categorías etiquetadas, que permite el tener que realizar el ATNLC sólo una vez

por cada categoría y que, aunque no llega a alcanzar los resultados del ATNLC, sí que mejora considerablemente los de otras soluciones (no cíclicas) de la literatura para obtener la invarianza al punto inicial (Sección 8.3.2).

- Una solución óptima basada en un pseudo-alineamiento (Sección 6.2), que sobre todo mejora los tiempos de clasificación (8.4.2).
- Una solución subóptima basada en AESA, pero que en la práctica se comporta bien y ofrece resultados similares a la solución correcta. Es muy improbable (véase la Sección 6.3.3 y los experimentos en la Sección 8.4.1) el encontrar una violación de la desigualdad triangular cuando tenemos descriptores de puntos con información global, es decir, elementos de cadena de varias dimensiones. Los experimentos con esta solución muestran que la mejora de tiempos es drástica, en clasificación y en recuperación de formas (Sección 8.4.2).
- El algoritmo LAESAEA (Sección 6.3.1), que como se muestra en los experimentos mejora los tiempos del LAESA (Sección 8.4.2).

Nótese que estas dos últimas propuestas también tienen cabida en otros contextos, no sólo en el de las cadenas cíclicas.

Con los modelos ocultos de Markov:

- Hemos argumentado (véase la Sección 7.1) y demostrado empíricamente que otras propuestas de la literatura para resolver el problema de la invarianza al punto inicial no parecen una buena solución y que nuestro heurístico (APR) mejora sus resultados en la mayoría de las ocasiones (Sección 8.5.1).
- Hemos demostrado experimentalmente que la topología izquierda-derecha lineal es suficiente para reconocer contornos (Sección 8.5.2), y con esto, podemos aprovechar el algoritmo cíclico eficiente propuesto para este tipo de topologías (Sección 7.5).
- Hemos formalizado la clasificación y aprendizaje cíclicos, formulando los algoritmos Baum-Welch cíclico y Viterbi cíclico (Sección 7.3). Hemos demostrado que este tratamiento cíclico es la mejor solución para conseguir la invarianza al punto inicial (véanse las Secciones 8.5.3 y 8.5.4).

9.2. Líneas futuras de investigación

Las principales líneas de investigación que se abren tras la realización de esta tesis se pueden resumir en las siguientes:

- Desarrollar algoritmos para el tratamiento de árboles de cadenas cíclicas [Bil03] que modelen algo más que contornos externos, como por ejemplo, huecos dentro de las figuras que podrían ser importantes para la discriminación.
- Continuar estudiando la correspondencia entre la dimensionalidad intrínseca y el incumplimiento de la desigualdad triangular, con el ATNL y AESA, para encontrar una mejor formalización y aplicarlo a otros contextos.

- Estudiar la posible reducción de la dimensionalidad de las cadenas con técnicas de cuantificación vectorial en sus descriptores de contorno.
- Extender el algoritmo divide y vencerás a otro tipo de topologías de modelos ocultos de Markov en el caso de que fueran necesarias.
- Estudiar otros tipos de modelos gráficos dirigidos que actualmente están en auge, como los modelos ocultos de Markov factoriales [GJ97] o los paralelos [VSM00, CLZ+09].

9.3. Publicaciones relacionadas con la tesis

Gran parte de las ideas desarrolladas en la presente tesis han sido presentadas en diversas publicaciones de ámbito internacional:

- Víctor M. Jiménez, Andrés Marzal, Vicente Palazón y Guillermo Peris, *Computing the Cyclic Edit Distance for Pattern Classification by Ranking Edit Paths*, Proc. Structural, Syntactic, and Statistical Pattern Recognition, LNCS, pp. 125-133, 2004.
- Andrés Marzal y Vicente Palazón, *Dynamic Time Warping of Cyclic Strings for Shape Matching*, Proc. Int. Conf. on Advances in Pattern Recognition, LNCS, pp. 644-652, 2005.
- Andrés Marzal, Vicente Palazón y Guillermo Peris, *Contour-Based Shape Retrieval Using Dynamic Time Warping*, Proc. Conf. of Spanish Association for Artificial Intelligence, LNCS, pp. 190-199, 2005.
- Andrés Marzal, Vicente Palazón y Guillermo Peris, *Shape Retrieval Using Normalized Fourier Descriptors Based Signatures and Cyclic Dynamic Time Warping*, Proc. Structural, Syntactic, and Statistical Pattern Recognition, LNCS, pp. 208-216, 2006.
- Vicente Palazón y Andrés Marzal, *Shape Retrieval Using Shape Contexts and Cyclic Dynamic Time Warping*, Proc. Int. Conf. on Image Analysis and Recognition, LNCS, pp. 624-635, 2006.
- Vicente Palazón y Andrés Marzal, *Speeding Up Shape Classification by Means of a Cyclic Dynamic Time Warping Lower Bound*, Proc. Int. Conf. Intelligent Data Engineering and Automated Learning, LNCS, pp. 436-443, 2006.
- Vicente Palazón y Andrés Marzal, *Cyclic Viterbi Score for Linear Hidden Markov Models*, Proc. Iberian Conference on Pattern Recognition and Image Analysis , LNCS, pp. 339-346, 2007.
- Vicente Palazón, Andrés Marzal y J. M. Vilar, *Cyclic Linear Hidden Markov Models for Shape Classification*, Proc. Pacific-Rim Symposium on Image and Video Technology, LNCS, pp. 152-165, 2007.

Bibliografía

[ABB95] H. Alt, B. Behrends, and J. Blomer. Approximate matching of polygonal shapes. *Annals of Mathematics and Artificial Intelligence*, 13(3-4):251–265, 1995.

[AC01] Aach and Church. Aligning gene expression time series with time warping algorithms. *BIOINF: Bioinformatics*, 17, 2001.

[AcH+91] Arkin, chew, Huttenlocher, Kedem, and Mitchell. An efficiently computable metric for comparing polygonal shapes. *IEEETPAMI: IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13, 1991.

[AD97] Gady Agam and Its'hak Dinstein. Geometric separation of partially overlapping nonrigid objects applied to automatic chromosome classification. *IEEE Trans. Pattern Anal. Mach. Intell*, 19(11):1212–1222, 1997.

[AFS93] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. *Lecture Notes in Computer Science*, pages 69–69, 1993.

[AG88] F. Ashby and R. Gott. Decision rules in the perception and categorization of multidimensional stimuli. *Journal of Experimental Psycology*, 14(1):33–53, 1988.

[AM01] Sadegh Abbasi and Farzin Mokhtarian. Affine-similar shape retrieval: application to multiview 3-D object recognition. *IEEE Transactions on Image Processing*, 10(1):131–139, 2001.

[AMK99] Sadegh Abbasi, Farzin Mokhtarian, and Josef Kittler. Curvature scale space image in shape similarity retrieval. *Multimedia Syst*, 7(6):467–476, 1999.

[AO03] Tomasz Adamek and Noel E. O'Connor. Efficient contour-based shape representation and matching. In Nicu Sebe, Michael S. Lew, and Chabane Djeraba, editors, *Proceedings of the 5th ACM SIGMM International Workshop on Multimedia Information Retrieval, MIR 2003, November 7, 2003, Berkeley, CA, USA*, pages 138–143. ACM, 2003.

[AO04] Tomasz Adamek and Noel E. O'Connor. A multiscale representation method for nonrigid shapes with a single closed contour. *IEEE Trans. Circuits Syst. Video Techn*, 14(5):742–753, 2004.

[ARKF07] N. Alajlan, I. El Rube, M. S. Kamel, and G. Freeman. Shape retrieval using triangle-area representation and dynamic space warping. *Pattern Recognition*, 40(7):1911–1920, July 2007.

[Att54] F. Attneave. Some informational aspects of visual perception. *Psychological Review*, 61:pp.183–193, 1954.

[AU73] Aho, A. V. and Ullman, J. D. *The Theory of Parsing, Translation and Compiling*. Prentice-Hall, 1973.

[AW99] R. Alferez and Y.F. Wang. Geometric and illumination invariants for object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(6):505–536, 1999.

[AYV00] N. Arica and F. Yarman-Vural. A shape descriptor based on circular hidden markov model. In *International Conference on Pattern Recognition*, pages Vol I: 924–927, 2000.

[AYV03] Nafiz Arica and Fatos T. Yarman-Vural. BAS: a perceptual shape descriptor based on the beam angle statistics. *Pattern Recognition Letters*, 24(9-10):1627–1639, 2003.

[Bau72] L. E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of a markov process. *Inequalities*, 3:1–8, 1972.

[BB93] H. Bunke and H. Bühler. Applications of approximate string matching to 2D shape recognition. *Pattern Recognition*, 26(12):1797–1812, 1993.

[BCGJ95] R. Basri, L. Costa, D. Geiger, and D. W. Jacobs. Determining the similarity of deformable shapes. In *Physics Based Modeling Workshop in Computer Vision*, page SESSION 5, 1995.

[BCP05] Ilaria Bartolini, Paolo Ciaccia, and Marco Patella. WARP: Accurate retrieval of shapes using phase of fourier descriptors and time warping distance. *IEEE Trans. Pattern Anal. Mach. Intell*, 27(1):142–147, 2005.

[BE67] L. E. Baum and J. A. Eagon. An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model of ecology. *Bull. Amer. Math. Soc.*, 73:360–363, 1967.

[Bil03] P. Bille. Tree edit distance, alignment distance and inclusion. *IT Univ. of Copenhagen TR-2003-23*, 2003.

[Bil05] P. Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337(1-3):217–239, 2005.

[Bis06] C. M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[BM04] Manuele Bicego and Vittorio Murino. Investigating hidden markov models' capabilities in 2D shape cassification. *IEEE Trans. Pattern Anal. Mach. Intell*, 26(2):281–286, 2004.

[BMF04] Bicego, Murino, and Figueiredo. Similarity-based classification of sequences using hidden markov models. *PATREC: Pattern Recognition, Pergamon Press*, 37, 2004.

[BMP02] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Anal. Mach. Intell*, 24(4):509–522, 2002.

[Bob01] Miroslaw Bober. MPEG-7 visual shape descriptors. *IEEE Trans. Circuits Syst. Video Techn*, 11(6):716–719, 2001.

[BPSW70] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *Ann. Math. Stat.*, 41:164–171, 1970.

[BS68] L.E. Baum and GR Sell. Growth functions for transformations on manifolds. *Pac. J. Math*, 27(2):211–227, 1968.

[Car01] M. Carlin. Measuring the performance of shape similarity retrieval methods. *Computer Vision and Image Understanding*, 1(84):44–61, 2001.

[CC96] W. S. Chan and F. Chin. On approximation of polygonal curves with minimum number of line segments or minimum error. *International Journal of Computational Geometry and Applications*, 6:59–77, 1996.

[CFSV04] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty years of graph matching in pattern recognition. *IJPRAI*, 18(3):265–298, 2004.

[CGK03] Antonio Cardone, Satyandra K. Gupta, and Mukul Karnik. A survey of shape similarity assessment algorithms for product design and manufacturing applications. *J. Comput. Inf. Sci. Eng*, 3(2):109–118, 2003.

[CK96] G. C. H. Chuang and C. C. J. Kuo. Wavelet descriptor of planar curves: Theory and applications. *IEEE Trans. Image Processing*, 5(1):56–70, January 1996.

[CL01] Jinhai Cai and Zhi-Qiang Liu. Hidden markov models with spectral features for 2D shape recognition. *IEEE Trans. Pattern Anal. Mach. Intell*, 23(12):1454–1458, 2001.

[CLR90] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press Cambridge, MA, 1990.

[CLZ+09] Changhong Chen, Jimin Liang, Heng Zhao, Haihong Hu, and Jie Tian. Factorial hmm and parallel hmm for gait recognition. *Trans. Sys. Man Cyber Part C*, 39(1):114–123, 2009.

[CNB98] M.S. Crouse, R.D. Nowak, and R.G. Baraniuk. Wavelet-based statistical signal processing using hidden Markov models. *IEEE transactions on signal processing*, 46(4):886–902, 1998.

[CNBYM01] E. Chávez, G. Navarro, R. Baeza-Yates, and J.L. Marroquín. Searching in metric spaces. *ACM Computing Surveys (CSUR)*, 33(3):273–321, 2001.

[CVR87] F. Casacuberta, E. Vidal, and H. Rulot. On the metric properties of dynamic time warping. *IEEE Trans. Acoustics, Speech and Signal Processing*, ASSP-35(11):1631, 1987.

[dBDLP98] A. del Bimbo, M. Demarsico, S. Levialdi, and G. Peritore. Query by dialog: An interactive approach to pictorial querying. *Image and Vision Computing*, 16(8):557–569, June 1998.

[DEKM98] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis*. Cambridge Univ. Press, 1998.

[DH73] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.

[DHS01] Richard Duda, Peter Hart, and David Stork. *Pattern Classification*. John Wiley and Sons, 2001. 0-471-05669-3.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society (Series B)*, 39(1):1–38, 1977.

[EKR98] S. Eickeler, A. Kosmala, and G. Rigoll. Hidden Markov Model based continuous online gesture recognition. In *International Conference on Pattern Recognition*, volume 14, pages 1206–1208. Citeseer, 1998.

[ERAK+05] I. El Rube, N. Alajlan, M. Kamel, M. Ahmed, and G. Freeman. Efficient multiscale shape-based representation and retrieval. *Lecture notes in computer science*, 3656:415, 2005.

[ERKA04] I. El Rube, M. Kamel, and M. Ahmed. 2-d shape matching using asymmetric wavelet-based dissimilarity measure. *Lecture Notes in Computer Science*, pages 368–375, 2004.

[FMJ97] A. Fred, J. Marques, and P. Jorge. Hidden markov models vs syntactic modeling in object recognition. In *ICIP97*, volume I, pages 893–896, 1997.

[For73] G. D. Forney. The Viterbi algorithm. *Proceedings of the IEEE*, 61:268–278, March 1973.

[Fre74] Herbert Freeman. Computer processing of line-drawing images. *ACM Computing Surveys*, 6(1):57–97, March 1974.

[FS02] Andre Folkers and Hanan Samet. Content-based image retrieval using fourier descriptors on a logo database. In *ICPR (3)*, pages 521–524, 2002.

[Fu82] K. S. Fu. *Syntactic pattern recognition and applications*. Prentice Hall, 1982.

[FvD96] J. Foley and A. van Dam. *Computer Graphics: Principles and Practice*. Addison-Wesley, 1996.

[GD95] D. M. Gavrila and L. S. Davis. Towards 3-D model-based tracking and recognition of human movement. In Martin Bichsel, editor, *Int. Workshop on Face and Gesture Recognition*, pages 272–277. IEEE Computer Society, June 1995.

[GJ97] Z. Ghahramani and M.I. Jordan. Factorial hidden Markov models. *Machine learning*, 29(2):245–273, 1997.

[GT78] R. C. Gonzalez and M. G. Thomason. *Syntactic Pattern Recognition: An Introduction*. Addison-Wesley, 1978.

[GW92] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Addison-Wesley, 3rd edition, 1992.

[GW99] Yoram Gdalyahu and Daphna Weinshall. Flexible syntactic matching of curves and its application to automatic hierarchical classification of silhouettes. *IEEE Trans. Pattern Anal. Mach. Intell*, 21(12):1312–1328, 1999.

[GXTL] Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis & Applications*.

[HBT96] J. Hu, M.K. Brown, and W. Turin. HMM based on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(10):1039–1045, 1996.

[HD80] P.A.V. Hall and G.R. Dowling. Approximate string matching. *ACM Computing Surveys (CSUR)*, 12(4):402, 1980.

[Het] M.L. Hetland. The Basic Principles of Metric Indexing.

[HK91] Y. He and A. Kundu. 2-D shape classification using hidden markov model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(11):1172–1184, November 1991.

[HK96] R. Hughey and A. Krogh. Hidden Markov models for sequence analysis: extension and analysis of the basic method. *Bioinformatics*, 12(2):95, 1996.

[HL02] Horng and Li. An automatic and efficient dynamic programming algorithm for polygonal approximation of digital curves. *PRL: Pattern Recognition Letters*, 23, 2002.

[Hor86] Berthold Klaus Paul Horn. *Robot Vision.* MIT Press, Cambridge, Massachusetts, 1986.

[Hu62] M. K. Hu. Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory*, 8:179–197, 1962.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages, and computation.* Addison Wesley, 1979.

[Ita75] F. Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Trans. Acoustics, Speech and Signal Processing*, ASSP-23(1):67–72, February 1975.

[Jäh01] B. Jähne. *Digital Image Processing.* Springer-Verlag, Berlin, 5th edition, 2001.

[Jai89] Anil K. Jain. *Fundamentals of Digital Image Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[JdFC95] Roberto Marcondes Cesar Junior and Luciano da Fontoura Costa. Piecewise linear segmentation of digital contours in O(N. log(N)) through a technique based on effective digital curvature estimation. *Real-Time Imaging*, 1(6):409–417, 1995.

[JKS95] R. C. Jain, R. Kasturi, and B. G. Schunck. *Machine Vision.* McGraw-Hill, 1995.

[JM03] J. R. Rico Juan and L. Mico. Comparison of AESA and LAESA search algorithms using string and tree-edit-distances. *Pattern Recognition Letters*, 24(9-10):1417–1426, June 2003.

[JMPP04] Víctor M. Jiménez, Andrés Marzal, Vicente Palazón, and Guillermo Peris. Computing the cyclic edit distance for pattern classification by ranking edit paths. In Ana L. N. Fred, Terry Caelli, Robert P. W. Duin, Aurélio C. Campilho, and Dick de Ridder, editors, *SSPR/SPR*, volume 3138 of *Lecture Notes in Computer Science*, pages 125–133. Springer, 2004.

[JP98] T. Jebara and A. Pentland. Action reaction learning: Automatic visual analysis and synthesis of interactive behaviour. *Lecture notes in computer science*, pages 273–292, 1998.

[JR90] B. H. Juang and L. R. Rabiner. The segmental K-means algorithm for estimating parameters of hidden markov models. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(9):1639, 1990.

[JZ97] AK Jain and D. Zongker. Representation and recognition of handwritten digits usingdeformable templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(12):1386–1390, 1997.

[KB02] M.I. Khalil and M.M. Bayoumi. Affine invariants for object recognition using the wavelet transform. *Pattern recognition letters*, 23(1-3):57–72, 2002.

[KBKNP96] R.S. Kashi, P. Bhoj-Kavde, R.S. Nowakowski, and T.V. Papathomas. 2-D shape representation and averaging using normalized wavelet descriptors. *Simulation*, 66(3):164, 1996.

[Keo02] Eamonn Keogh. Exact indexing of dynamic time warping. In *Proceedings 28th International Conference on Very Large Databases, Hong Kong*, pages 406–417, December 2002.

[KF07] A. Kolesnikov and P. Franti. Polygonal approximation of closed discrete curves. *Pattern Recognition*, 40(4):1282–1293, April 2007.

[Kin03] V. V. Kindratenko. On using functions to describe the shape. *J. Math. Imaging and Vision*, 18:225–245, 2003.

[Knu93] Donald Ervin Knuth. *The Stanford Graphbase: A Platform for Combinatorial Computing*. ACM Press, 1993.

[KPZ+04] E. Keogh, T. Palpanas, V.B. Zordan, D. Gunopulos, and M. Cardle. Indexing large human-motion databases. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 780–791. VLDB Endowment, 2004.

[KR05] Eamonn J. Keogh and Chotirat (Ann) Ratanamahatana. Exact indexing of dynamic time warping. *Knowl. Inf. Syst*, 7(3):358–386, 2005.

[KV00] Zsolt Miklós Kovács-Vajna. A fingerprint verification system based on triangular matching and dynamic time warping. *IEEE Trans. Pattern Anal. Mach. Intell*, 22(11):1266–1276, 2000.

[KWX+06] E. Keogh, L. Wei, X. Xi, S. Lee, and M. Vlachos. LB Keogh supports exact indexing of shapes under rotation invariance with arbitrary representations and distance measures. *VLDB, 2006*, pages 882–893, 2006.

[Lee89] K.F. Lee. *Automatic speech recognition: the development of the SPHINX system*. Kluwer Academic Pub, 1989.

[Lem09] D. Lemire. Faster retrieval with a two-pass dynamic-time-warping lower bound. *Pattern Recognition*, 42(9):2169–2180, 2009.

[Lev66] VI Levenshteiti. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics-Doklady*, volume 10, 1966.

[LL00] Longin Jan Latecki and Rolf Lakämper. Shape similarity measure based on correspondence of visual parts. *IEEE Trans. Pattern Anal. Mach. Intell*, 22(10):1185–1190, 2000.

[LLE00] L. Latecki, R. Lakämper, and U. Eckhardt. Shape descriptors for non-rigid shapes with a single closed contour. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR-00)*, pages 424–429, Los Alamitos, June 13–15 2000. IEEE.

[Lon98] S. Loncaric. A survey of shape analysis techniques. *Pattern Recognition*, 31(8):983–1001, 1998.

[LS90] Hong-Chih Liu and Mandyam D. Srinath. Partial shape classification using contour matching in distance transformation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-12(11):1072–1079, November 1990.

[Mae90] M. Maes. On a cyclic string-to-string correction problem. *Information Processing Letters*, 35:73–78, 1990.

[Mae91] Maurice Maes. Polygonal shape recognition using string-matching techniques. *Pattern Recognition*, 24(5):433–440, 1991.

[Mar76] David Marr. Early processing of visual information. In *Proceedings of the Royal Society*, volume 275 B, pages 483–519, 1976.

[MB00] A. Marzal and S. Barrachina. Speeding up the computation of the edit distance for cyclic strings. *Int. Conf. on Pattern Recognition*, pages 271–280, 2000.

[MB03] F. Mokhtarian and M. Bober. *Curvature Scale Space Representation: Theory, Applications, and MPEG-7 Standardization.* Kluwer, August 2003.

[MG01] A. Marzal and G.Peris. Dependencia del cálculo de la distancia de edición cíclica con las funciones de coste. Technical report, Universitat Jaume I, 2001.

[Mic96] L. Micó. *Algoritmos de búsqueda de vecinos más próximos en espacios métricos.* PhD thesis, 1996.

[MKA] F. Mokhtarian, J. Kittler, and S. Abbasi. Shape queries using image databases. http://www.ee.surrey.ac.uk/Research/VSSP/imagedb/demo.html.

[MM86] F. Mokhtarian and A. Mackworth. Scale based description and recognition of planar curves and two-dimensiona shapes. *IEEE Trans. Pattern Analysis and Machine Intelligence*, PAMI-8(1), January 1986.

[MM92] F. Mokhtarian and A. K. Mackworth. A theory of multiscale, curvature-based shape representation for planar curves. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 14(8):789–805, August 1992.

[MMS+01] H. Muller, W. Muller, D.M. Squire, S. Marchand-Maillet, and T. Pun. Performance evaluation in content-based immage retrieval: overview and proposals. *Pattern Recognition Letters*, 5(22):593–601, 2001.

[MOV94] M.L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements. *Pattern Recognition Letters*, 15(1):9–17, 1994.

[MP80] Masek and Paterson. A faster algorithm computing string edit distances. *JCSS: Journal of Computer and System Sciences*, 20, 1980.

[MP98] E. E. Milios and E. G. M. Petrakis. Efficient shape matching and retrieval at multiple scales. In *Groningen Image Processing System, GIPSY*, 1998.

[MP99] Mario E. Munich and Pietro Perona. Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification. In *ICCV*, pages 108–115, 1999.

[MRR80] C. S. Meyers, L. R. Rabiner, and A. E. Rosenberg. Performance tradeoffs in dynamic time warping algorithms for isolated word recognition. *ieeeassp*, 28(6):623–635, Dec., 1980.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[MSMO03] F. Moreno-Seco, L. Mico, and J. Oncina. A modification of the LAESA algorithm for approximated k-NN classification. *Pattern Recognition Letters*, 24(1-3):47–53, 2003.

[Mun57] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, March 1957.

[MV93] A. Marzal and E. Vidal. Computation of normalized edit distance and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15, 1993.

[Nav01] G. Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):88, 2001.

[NB06] Neuhaus and Bunke. Edit distance-based kernel functions for structural pattern classification. *PATREC: Pattern Recognition, Pergamon Press*, 39, 2006.

[PF77] Eric Persoon and King-Sun Fu. Shape discrimination using fourier descriptors. *Trans. IEEE*, SMC-7(3):170–179, 1977.

[Pil06] Robert A. Pilgrim. Munkres' assignment algorithm. Modified for rectangular matrices. Online at http://csclab.murraystate.edu/bob.pilgrim/445/munkres.html, April 2006. Accessed 2006-07-18.

[PS00] R. Plamondon and S.N. Srihari. On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 63–84, 2000.

[PV94] J. C. Perez and E. Vidal. Optimum polygonal-approximation of digitized-curves. *Pattern Recognition Letters*, 15(8):743–750, August 1994.

[Rab89] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc IEEE*, 77(2), 1989.

[Rij79] C. J. Rijsbergen. *Information Retrieval*. London: Butterworth, 1979.

[RJ93] L. Rabiner and B-H. Juang. *Fundamentals of Speech Recognition*. Prentice-Hall, 1993.

[RK04] Chotirat (Ann) Ratanamahatana and Eamonn J. Keogh. Everything you know about dynamic time warping is wrong. In *KDD*, 2004.

[Sai00] N. Saito. Local feature extraction and its applications using a library of bases. *Topics in Analysis and Its Applications: Selected Theses*, pages 269–451, 2000.

[Sal01] M. Salotti. An efficient algorithm for the optimal polygonal approximation of digitized curves. *Pattern Recognition Letters*, 22(2):215–221, February 2001.

[SB85] M. Shridhar and A. Badreldin. A high-accuracy syntactic recognition algorithm for handwritten numerals. *IEEE Trans. Systems, Man and Cybernetics*, 15:152–158, 1985.

[SC78] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *ieeeassp*, 26(1):43–49, Feb., 1978.

[Sch78] G. Schwarz. Estimating the dimension of a model. *Ann. Stat.*, 14:461–64, 1978.

[SCTK98] D. Sharvit, J. Chan, H. Tek, and B. B. Kimia. Symmetry-based indexing of image databases. In *Workshop on Content-Based Access of Image and Video Libraries*, pages 56–62, 1998.

[Sel80] Peter H. Sellers. The theory and computation of evolutionary distances: Pattern recognition. *Journal of Algorithms*, 1(4):359–373, December 1980.

[SF81] J. A. Saghri and H. Freeman. Analysis of the precision of generalized chain codes for the representation of planar curves. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 3(5):533–539, September 1981.

[SK83] D. Sankoff and J. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison*. Addison-Wesley, Reading, MA, 1983.

[SKK03] T. B. Sebastian, P. N. Klein, and B. B. Kimia. On aligning curves. *IEEE Trans. Pattern Anal. Machine Intelligence*, 25:116–124, 2003.

[SMC05] Yutao Shou, Nikos Mamoulis, and David W. Cheung. Fast and exact warping of time series using adaptive segmental approximations. *Machine Learning*, 58(2-3):231–267, 2005.

[TB95] Q.M. Tieng and W.W. Boles. An application of wavelet-based affine-invariant representation. *Pattern Recognition Letters*, 16(12):1287–1296, 1995.

[TGJ07] N. Thakoor, J. Gao, and S. Jung. Hidden markov model-based weighted likelihood discriminant for 2-D shape classification. *IEEE Trans. Image Processing*, 16(11):2707–2719, November 2007.

[Tub89] JD Tubbs. A note on binary template matching. *Pattern Recognition*, 22(4):359–365, 1989.

[Tve77] A. Tversky. Features of similarity. *Psychological Review*, 84(4):327–352, 1977.

[UBS02] C. Urdiales, A. Bandera, and F. Sandoval. Non-parametric planar shape representation based on adaptive curvature functions. *Pattern Recognition*, 35(1):43–53, January 2002.

[VCBL88] E. Vidal, F. Casacuberta, J. Benedi, and M. Lloret. On the verification of triangle inequality by dynamic time-warping dissimilarity measures. *Speech Commun.*, 7(1):67–79, 1988.

[VCR85] E. Vidal, F. Casacuberta, and H. M. Rulot. Is the DTW distance really a metric? An algorithm reducing the number of DTW comparisons in isolated word recognition. *Speech Communication*, 4(4):333–344, 1985.

[VH01] R. C. Veltkamp and M. Hagedoorn. State-of-the-art in shape matching, February 20 2001.

[VHGK03] Michail Vlachos, Marios Hadjieleftheriou, Dimitrios Gunopulos, and Eamonn Keogh. Indexing multi-dimensional time-series with support for multiple distance measures. In Pedro Domingos, Christos Faloutsos, Ted SEnator, Hillol Kargupta, and Lise Getoor, editors, *Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-03)*, pages 216–225, New York, August 24–27 2003. ACM Press.

[Vid86] E. Vidal. An algorithm for finding nearest neighbours in (approximately) constant average time. *Pattern Recognition Letters*, 4(3):145–157, 1986.

[Vid94] E. Vidal. New formulation and improvements of the nearest-neighbour approximating and eliminating search algorithm (AESA). *Pattern Recognition Letters*, 15(1):1–7, 1994.

[Vit67] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Trans. Information Theory*, IT-13:260–269, April 1967.

[VLM06] I.F. Vega-Lopez and B. Moon. Quantizing time series for efficient similarity search under time warping. *ACST*, 6:334–339, 2006.

[VMA95] E. Vidal, A. Marzal, and P. Aibar. Fast computation of normalized edit distances. *IEEETPAMI: IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17, 1995.

[VRCB88] E. Vidal, H. M. Rulot, F. Casacuberta, and J. M. Benedi. On the use of a metric-space search algorithm (AESA) for fast DTW-based recognition of isolated words. *IEEE Trans. Acoustics, Speech and Signal Processing*, ASSP-36(5):651, 1988.

[VSM00] C. Vogler, H. Sun, and D. Metaxas. A framework for motion recognition with applications to American sign language and gait recognition. In *IEEE Workshop on Human Motion*. Citeseer, 2000.

[WB99] A.D. Wilson and A.F. Bobick. Parametric hidden markov models for gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1999.

[WF74] R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

[WSB98] R. Weber, H.J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the International Conference on Very Large Data Bases*, pages 194–205. INSTITUTE OF ELECTRICAL & ELECTRONICS ENGINEERS, 1998.

[Wu83] C.F.J. Wu. On the convergence properties of the EM algorithm. *The Annals of Statistics*, pages 95–103, 1983.

[YJF98] B.K. Yi, HV Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proceedings of the International Conference on Data Engineering*, pages 201–208. INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, 1998.

[YOO+95] S. Young, J. Odell, D. Ollason, V. Valtchev, and P. Woodland. The HTK Book. *Cambridge University*, 1996, 1995.

[Zan69] W. I. Zangwill. *Nonlinear Programming. A Unified Approach*. Prentice-Hall, Englewood Cliffs, NJ, 1969.

[ZL02] D. Zhang and G. Lu. A comparative study of curvature scale space and fourier descriptors for shape-based image retrieval. *Journal of Visual Communication and Image Representation*, 14(1):39–57, March 2002.

[ZL04] D. Zhang and G. Lu. Review of shape representation and description techniques. *Pattern Recognition*, 37:1–19, 2004.

[ZL05] Dengsheng Zhang and Guojun Lu. Study and evaluation of different fourier methods for image retrieval. *Image Vision Comput*, 23(1):33–49, 2005.

[ZS03] Yunyue Zhu and Dennis Shasha. Warping indexes with envelope transforms for query by humming. In ACM, editor, *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data 2003, San Diego, California, June 09–12, 2003*, pages 181–192, pub-ACM:adr, 2003. ACM Press.