INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

CÓMPUTO DE LA SIMILITUD ENTRE FIGURAS GEOMÉTRICAS

T E S I S

QUE PARA OBTENER EL GRADO DE MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A

MARIO ALBERTO ANGELES YRETA

DIRECTOR DE TESIS
DR. JESUS G. FIGUEROA NAZUNO

MÉXICO D. F. Junio 2006

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

En este trabajo se aborda el problema de determinar *similitud o semejanza entre figuras geométricas*; la importancia de dicho problema radica en su vasta aplicabilidad. Aunque el humano es capaz de *ligar* entidades distintas (personas, situaciones políticas, etc.) en base a su similitud/disimilitud, con *aparente* facilidad, los mecanismos empleados aún son desconocidos.

La dificultad de encontrar un método cuyo *criterio de comparación* sea *universalmente* aceptado aún en un contexto particular (imágenes, secuencias, objetos 3D, ontologías, códigos fuente, textos, etc.), hace que el problema de semejanza sea el foco de atención para muchos investigadores.

En este trabajo se presentan dos métodos para *representar y buscar similitud* entre figuras geométricas. Por representación se refiere a la codificación de las entidades (figuras geométricas), por medio de secuencias, dicha representación resulta conveniente para la detección de relaciones de similitud adimensional (*RNA Fuzzy ART*) y dimensional (*Dynamic Time Warping*).

El primer método para representar y buscar similitud entre figuras geométricas, no contempla etapas de extracción, selección, y/o construcción de características clásicas; dicho método utiliza una tripleta de secuencias acopladas para representar una figura dado, cada secuencia representa un solo eje cartesiano (*X*, *Y*, y *Z*); estas secuencias son presentadas a una tripleta de *RNA's Fuzzy ART*, éstas comparten los mismos valores de parámetros, y determinan la pertenencia de una figura a una clase utilizando la tripleta de categorías en forma de cadenas binarias que han sido generadas por las *RNA Fuzzy ART* y la *distancia de Hamming*.

El segundo método, representa figuras y su respectiva matriz de adyacencia, en *secuencias*; y determina distancias de semejanza usando *Dynamic Time Warping* (*DTW*). Aunque se ha probado la eficacia de *DTW* sobre la métrica Euclidiana, en este trabajo se exploran algunas de las propiedades de *DTW*, como los detalles de implementación y su complejidad temporal.

Son presentadas pruebas experimentales con figuras geométricas creadas con herramientas *DAC* (*Diseño Asistido por Computadora*), aplicando los métodos propuestos en este trabajo. Métodos y técnicas son detalladas y analizadas

cuidadosamente de forma aislada. Adicionalmente, se muestran dos técnicas para representar e indexar figuras en estructuras multidimensionales.

Los códigos utilizados en este trabajo son proporcionados en programas auto-contenidos, sin aspectos gráficos que puedan alejar al lector del problema de similitud. No es propósito del trabajo desarrollar una herramienta sino desarrollar ideas.

ABSTRACT

In this work the problem is approached to determine *similarity among polyhedrons*; the importance of this problem is in its vast applicability. Although the human is able to *bind* to different entities (people, political situations, etc.) on the basis of his similarity/dissimilarity, with *apparent* facility, the mechanisms still used are not known.

The difficulty to find a method whose *criterion of comparison* is universally accepted even in a particular context (images, sequences, objects 3D, ontologies, source codes, texts, etc.), makes that the similarity problem let be the center of attention for many researches.

In this work two methods are introduced to *represent and to search similarity* among polyhedrons. By representation we mean the codification of entities to compare, by means of sequences, this representation is advisable for the detection of dimensionless (*RNA Fuzzy ART*) and dimensional (*Dynamic Time Warping*) similarity relations.

The first method to represent and to determine similarity among polyhedrons, does not contemplate classic stages of feature extraction, selection, or construction; this method creates a triplet of *coupled sequences* to represent a single polyhedron, each sequence represent a single coordinate axis (*X*, *Y*, and *Z*); the triplet of coupled sequences is presented to a triplet of *ANN's Fuzzy ART*, these share the same parameters values, and determines the membership of a polyhedron to a class using the triplet of categories as binary strings that have been generated by the *ANN's Fuzzy ART* and the *Hamming distance*.

The second method represents polyhedrons and their respective adjency matrix in *sequences*; and determines distances of similarity using *Dynamic Time Warping* (*DTW*). Although have been proven the effectiveness of *DTW* over the Euclidean metric, in this work some of these properties are explored, like implementation details and time complexity.

Experimental tests with polyhedrons created with *CAD* tools (*Computer Assisted Design*) are presented, applying the methods introduced in this work are presented. Methods and techniques are detailed and analyzed carefully and in an isolated form.

Additionally, two techniques to represent and to index polyhedrons in multidimensional structures are presented.

The codes used in this work are provided in self-contained programs, without graphical aspects that can move away the interested one of the similarity problem. It is not intention of the work to develop a tool but ideas.

PRÓLOGO

OBJETIVO

El objetivo de este trabajo es presentar *dos* nuevos métodos para *representar y calcular similitud entre figuras geométricas*. Ello involucra:

1) La representación de entidades tridimensionales en *secuencias*. La creación de dichas secuencias no involucra descriptor alguno conocido en la literatura.

2) El cómputo de similitud *no supervisado*, de bajo costo computacional. Es decir, todos los algoritmos involucrados en los métodos de similitud deberán ser en tiempo y espacio *polinomial*.

ALCANCES

Aspectos filosóficos acerca del problema de similitud no son abordados, un enfoque práctico algorítmico, es establecido desde este momento.

Los objetos de estudio son figuras geométricas generadas por un sistema de diseño asistido por computadora (DAC) típico y figuras seudo-aleatorias, éstos pueden ser figuras geométricas abiertas.

Conocimiento general acerca de estructuras de datos multidimensionales como R-Tree y sus variantes es asumido, ya que el objetivo principal traza la línea que divide la búsqueda secuencial y la indexación de objetos, ésta última como una *aproximación* a la primera.

Este trabajo no es un estudio de técnicas de *extracción*, *selección*, y/o *construcción* de *características*, ya que los métodos propuestos no requieren de ninguna de ellas.

Este trabajo tampoco es un análisis de las diversas técnicas existentes para determinar similitud entre objetos 3D. El argumento es claro, no existe un criterio universalmente aceptado de similitud, a diferencia de otros problemas, como el de predicción, no hay un error mesurable. Aunque existen clasificaciones tácitamente aceptadas, por ejemplo, la taxonomía de los seres vivos, siempre son factibles a interpretaciones contradictorias.

MOTIVACIÓN

El problema de calcular similitud se ha manifestado como un problema no trivial, adicionalmente, ha mostrado la capacidad de proporcionar caminos alternos para la solución de otros problemas, como puede ser la predicción de series de tiempo, el problema de detectar relación entre entidades, específicamente encontrar *motivos*, etc.

La motivación principal para realizar este trabajo es la de proporcionar una *perspectiva* adicional al problema de similitud; no sólo métodos ó técnicas para determinar una relación de similitud entre figuras geométricas.

Otros motivos que impulsaron la realización de este trabajo de tesis, son los trabajos realizados por el *autor* y *asesor* en diferentes espacios sobre el problema de similitud, y su gran ubicuidad en la ciencia de la computación.

APORTACIONES

En el problema de similitud, al igual que en muchos otros problemas, el enfoque *divide y vencer* resulta de gran utilidad en aquellos casos en que podemos obtener un *atŏmum*, como parte indivisible y elemental de las entidades que comparamos. De forma contraria, los métodos que se proponen en este trabajo argumentan la ***posibilidad***, de utilizar una ***adecuada*** representación de las entidades y un ***adecuado*** criterio de

comparación como un ***todo***, sin la necesidad de determinar a ***priori*** las partes elementales de las entidades en cuestión.

Aunque el objetivo de este trabajo es presentar dos métodos para representar y determinar similitud entre figuras geométricas, se ***exploran*** algunas propiedades del problema de similitud. Concretamente, la ***estructura conceptual*** del problema de similitud.

De forma puntual las aportaciones de este trabajo son:

- ✓ Un nuevo método para representar y determinar similitud adimensional entre figuras geométricas, aplicando *RNAs Fuzzy ART*, dicho método ***no*** incluye etapas de extracción, selección, y/o construcción de características ***clásicas***. Este método es idóneo para figuras de tamaño ***similar***

- ✓ Un nuevo método para representar y determinar similitud entre figuras geométricas, aplicando *Dynamic Time Warping* y *Fast Dynamic Time Warping*, dicho método ***no*** incluye etapas de extracción, selección, y/o construcción de características ***clásicas***. Este método es idóneo para figuras de similar ó ***diferente*** tamaño.

- ✓ Ambos métodos pueden comparar figuras geométricas con caras ***abiertas*** (no sólidos).

- ✓ ***Proposición*** de una estructura conceptual del problema de similitud.

- ✓ La ***sugerencia*** de reducción de complejidad evitando en lo posible ***funciones recursivas***.

- ✓ Un enfoque adicional: “Representar entidades, en este caso figuras en secuencias ***representativas***, permite una eficiente comparación”.

- ✓ El problema de similitud es difícil de abordar en todos aquellos casos en los que las entidades no tienen un ***orden*** pre-establecido.

ESTRUCTURA DEL TRABAJO

En el capítulo 1 se introduce el *problema de similitud*, algunas definiciones, así como su *importancia*; el *estado del arte* se resumen en el capítulo 2.

En el capítulo 3 se propone una *estructura conceptual* al problema de similitud, la comprensión de capítulos posteriores no requiere la lectura de este capítulo.

En el capítulo 4 se detalla la primera técnica involucrada en este trabajo, la *RNA Fuzzy ART* como elemento indispensable para determinar similitud entre figuras geométricas por medio de secuencias acopladas, el objetivo de este capítulo es analizar cada paso del algoritmo de la *RNA Fuzzy ART*, así como sus características.

En el capítulo 5 se analiza el algoritmo de *Dynamic Time Warping (DTW)*. Este capítulo proporciona definiciones, estrategias de reducción de complejidad y concreta con una variante del algoritmo original llamado *Fast Dynamic Time Warping*, dicho algoritmo es usado en el segundo método de similitud propuesto en este trabajo.

En el capítulo 6, se especifican los detalles de creación del *conjunto de datos* utilizado en las pruebas experimentales cuyos *resultados* aparecen en el capítulo 9.

Los capítulos 7 y 8 introducen los dos métodos de *representación y cómputo de similitud entre figuras geométricas*, el primer método está basado en una tripleta de *RNA Fuzzy ART* (capítulo 7), el segundo, basa su criterio de semejanza en la función *DTW*.

Por último en el capítulo 10 se hace una breve *discusión* de los resultados obtenidos en las pruebas experimentales; y el capítulo 11 se presentan las *conclusiones* del presente trabajo.

1 INTRODUCCIÓN

Una de las capacidades más sorprendentes del ser humano es su capacidad de *abstracción*, consecuentemente, éstos logran *ligar* fenómenos que a primera instancia parecieran lejanos.

Los criterios y mecanismos que el ser humano emplea para determinar dicha relación son múltiples y en ocasiones desconocidos. Es así que relacionar entidades es un aspecto fundamental de la *inteligencia humana*.

El humano aborda el problema de similitud o semejanza de forma *aparentemente* simple, es capaz de reconocer el *parecido* entre diferentes entidades, como dos canciones similares, relatos que refieren a un mismo evento, las coincidencias políticas entre candidatos, las características que a su juicio hacen coincidentes a las personas, etc. Todo ello involucra procesos físicos dentro del cerebro humano que hasta la fecha son poco conocidos.

Los primeros esfuerzos para entender el problema de similitud los dio Tversky [Tversky77] en 1977, definió que problema de similitud se puede abordar en cuatro enfoques, ver figura 1;

a) El primero, refiere a la comparación de entidades usando como criterio los *elementos comunes* (*common elements approach*).

b) Otro enfoque es el de realizar comparaciones en donde se persigue *emparejar* una entidad con respecto a la otra (*template approach*).

c) Un tercer enfoque se refiere a las comparaciones de entidades por su *geometría* (*geometric approach*).

d) Por último, el enfoque de *características* (*feature approach*), este enfoque es similar a de *elementos comunes*, aunque éste último no requiere identificar a priori las propiedades involucradas en la comparación.

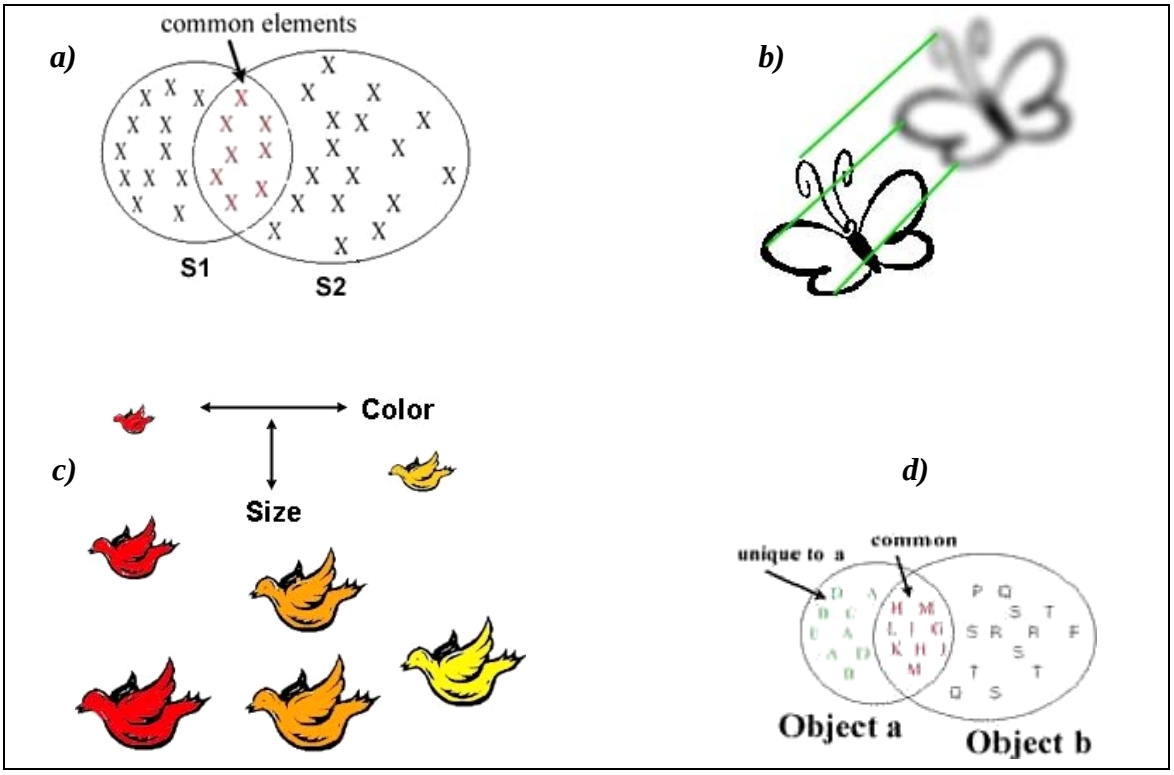


Figura 1. Enfoques del problema de similitud declarados por Tversky.

La apreciación de Tversky es que el problema de similitud tiene características propias, inherentes al contexto.

El último esfuerzo realizado para entender el problema de similitud lo dio Lin en [Lin98], la perspectiva de Lin es la conceptualizar el problema de similitud como un cociente entre las concordancias de las entidades involucradas y la descripción de éstas, dicho enfoque asume seis cosas:

1) La concordancia entre *A* y *B* es medida por (**1**)

$$I(common(A,B)) \qquad (1)$$

Donde *common*(*A*,*B*) es una proposición que establece las concordancias entre A y B, y *I*(*s*) es la cantidad de información contenida en la proposición *s*.

2) La diferencia entre *A* y *B* es medida por (**2**)

$$I(description(A,B)) - I(common(A,B)) \qquad (2)$$

Donde *description*(*A*,*B*) es una proposición que describe lo que son *A* y *B*.

3) La similitud entre A y B, sim(A,B), es una función de sus concordancias y diferencias (**3**). Esto es

$$\text{sim}(A,B) = f(\text{I}(common(A,B)), \text{I}(descripción(A,B))) \quad (\mathbf{3})$$

El dominio de f es $\{(x,y) | \, x \geq 0, y > 0, y \geq x\}$

4) La similitud entre un par de idénticos objetos es 1.

5) $\forall y > 0, f(0,y) = 0$.

6) Derivada de las anteriores.

Lin manipula los seis preceptos anteriores y demuestra que:

$$sim(A,B) = \frac{\log P(common(A,B))}{\log P(description(A,B))} \quad (\mathbf{4})$$

Existen al menos dos argumentos por los cuales la función de similitud expresada como un cociente, es una definición poco general sobre el problema de similitud. Estos aspectos son:

1) Asume un modelo probabilístico.

2) Limita el problema de similitud a mapeo de un dominio contra un contradominio no definido. Soslayando la posibilidad de utilizar diferentes sistemas de medida [Krantz71] [Suppes89] [Luce90].

El lector puede observar que el problema de similitud esta latente en muchas áreas de la computación, como es reconocimiento de patrones, reconocimiento de voz, minería de datos, entre otras.

Una característica que hace especial al problema de similitud es la dificultad de determinar alguna forma de error en los métodos de comparación. En la figura 2 se *ilustra* lo anterior.

El elemento esencial para estimar un error o desviación es conocer el *punto de referencia*, ello implica que exista de antemano ya sea una clasificación de las entidades a comparar, un orden preestablecido, o un contexto acotado.

Figura 2. El problema de similitud. Nótese que múltiples criterios de comparación pueden ser usados; volumen, forma geométrica, entre otras; y ninguno de ellos tiene que ser el único.

1.1 DEFINICIÓN DE SIMILITUD

Intencionalmente se ha reservado la definición etimológica de similitud. De acuerdo a la *Real Academia de la Lengua Española*, similitud se refiere a: *"La cualidad de ser semejante"*.

La similitud no se conceptualiza en base a un espacio métrico, ni en forma recursiva. Es por ello que en el capítulo 3 se *esboza* una *Estructura Conceptual del Problema de Similitud*, es decir, qué elementos intervienen en el problema de similitud y la relación entre ellos.

1.2 IMPORTANCIA DEL PROBLEMA DE SIMILITUD

El problema de similitud es tan útil como común. Conocer comportamientos parecidos de bolsas de valores, electrocardiogramas similares, huellas digitales similares, hábitos de compra similares, rostros similares, casas similares, perfiles políticos similares, textos similares (plagio), códigos fuente similares, etc.

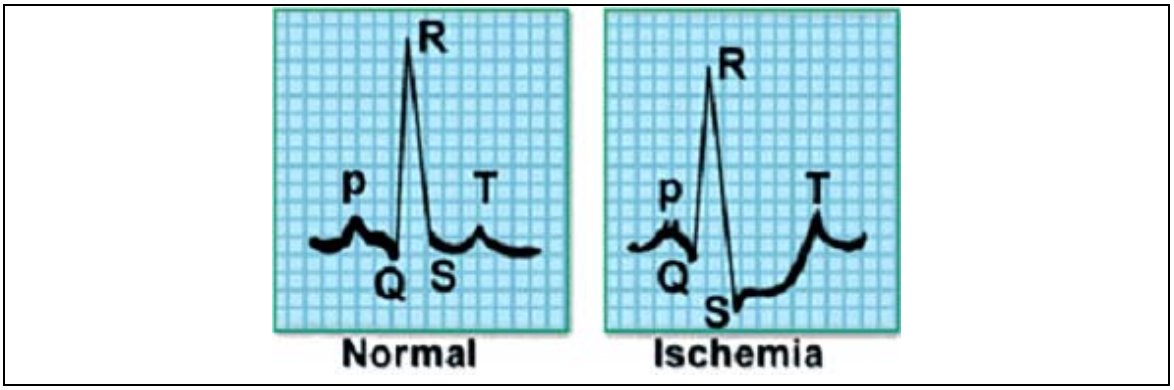


Figura 3. Búsqueda de similitud en electrocardiogramas. Aunque son conocidos los patrones significativos (P, Q, R, S, y T) de los electrocardiogramas, sigue siendo un problema abierto.

Es así que el problema de similitud se considera de gran utilidad científica, social y económica. Cuando se declara que el problema de similitud es *ubicuo* nos referimos a la capacidad de abstraer entidades y encontrar los mecanismos para su comparación.

Recientemente, se ha concebido al problema de similitud como una vía alterna a la solución de otros problemas, como lo es el de la predicción de series de tiempo.

En la figura 4 se muestran series de tiempo de indicadores económicos. Ya que en ocasiones la predicción resulta en un proceso poco confiable, se ha propuesto la posibilidad de encontrar fenómenos similares que contribuyan a una predicción confiable.

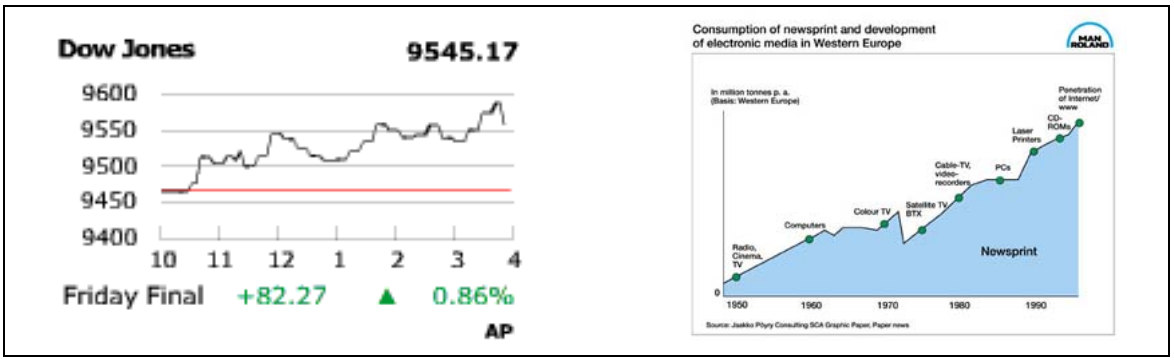


Figura 4. Ubicuidad del problema de similitud. A la izquierda se encuentra una serie de tiempo del índice Dow Jones, comúnmente usado en las bolsas de valores, a la derecha se encuentra la serie de tiempo del consumo de medio de impresos y desarrollo de medios electrónico en Europa Occidental.

Existen múltiples técnicas y métodos para determinar relación entre señales, series de tiempo, secuencias, arreglos, vectores, imágenes, matrices, ontologías, redes semánticas, textos.

Es de particular interés de este trabajo se abordar el problema de similitud entre **figuras geométricas**.

REFERENCIAS DEL CAPÍTULO

[Tversky77] Tversky A. "Features of Similarity". *In Readings in Cognitive Science*. Morgan Kaufmann 1988. (From Psychological Review 84, 327-352, 1977).

[Lin98] Lin D. "An information-theoretic definition of similarity". *In Proceedings of the 15th International Conference on Machine Learning*, pp. 196-304, 1998.

[Krantz71] Krantz, D. H., Luce, R. D., Suppes, P., & Tversky, A. (1971). *Foundations of measurement: Vol. 1. Additive and polynomial representations*. New York: Academic.

[Luce90] Luce, R. D., Krantz, D. H., Suppes, P., & Tversky, A. (1990). *Foundations of measurement: Vol. 3. Representation, axiomatization, and invariance*. San Diego, CA: Academic.

[Suppes89] Suppes, P., Krantz, D. H., Luce, R. D., & Tversky, A. (1989). *Foundations of measurement: Vol. 2. Geometrical, threshold, and probabilistic representations*. San Diego, CA: Academic.

2 ESTADO DEL ARTE

El problema de similitud entre figuras geométricas u objetos 3D ha sido abordado por varios investigadores alrededor del planeta. Sin embargo las ideas principales de los trabajos pioneros han permanecido.

Los primeros trabajos en que se vio envuelto el problema de similitud entre figuras geométricas fueron [Roberts65] y [Guzmán68], ver figura 5.

Guzmán introdujo la siguiente idea; existen elementos base, *atŏmum* (átomo), *cosas indivisibles*, que constituyen un patrón y éste a su vez, a una figura. Una vez que se tiene dicha *caracterización* de estas figuras, es posible determinar su clase.

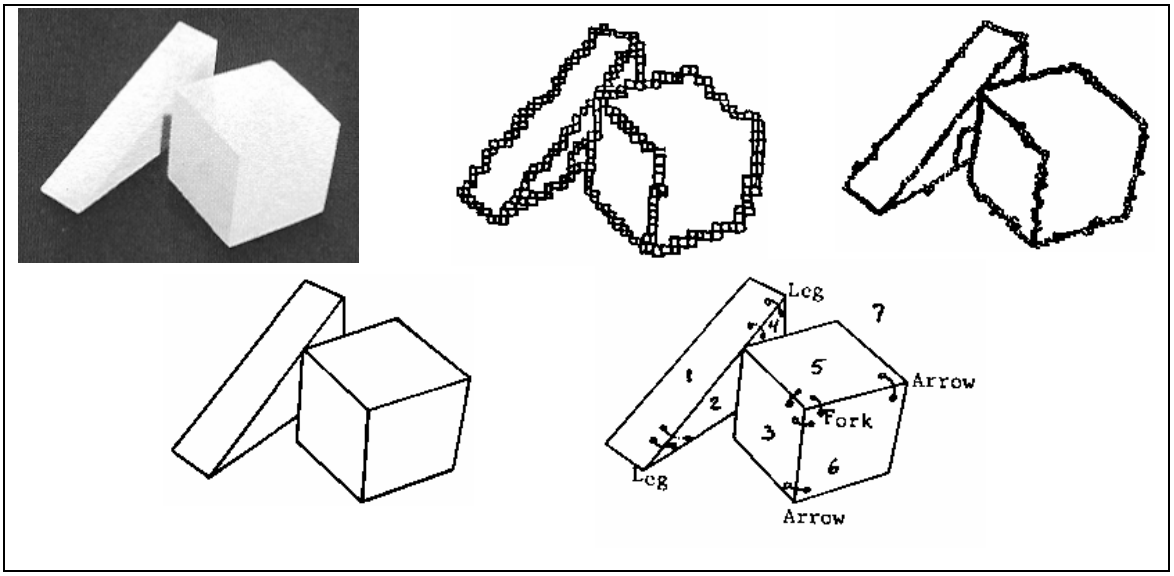


Figura 5. Reconocimiento de Figuras. Imágenes extraídas de [Guzmán68]

Es a partir de estos trabajos que se refuerza la idea de *dividir* y *vencer* en el área de reconocimiento de patrones, así como la idea de definir a priori ciertas premisas o características *relevantes* de un objeto dado.

Recientemente se presentó en [Shilane04] un estudio sobre 12 técnicas para calcular similitud entre figuras u objetos 3D. En el último párrafo del resumen dice: "*Based on*

experiments with several different shape descriptors, we conclude that no single descriptor is best for all classifications…"

Lo que el estudio refleja es la imposibilidad de un solo método de similitud sea universalmente aceptado, ya que cada uno de ellos determina el problema de similitud de acuerdo a sus necesidades.

En [Brennecke04] se presentó un método para determinar similitud entre *formas 3D*, la idea de este método es buscar el *subgrafo común más grande* (*biggest common subgraph*). Sabiendo que esto representa un problema *NP Completo*, progresivamente reducen el número de vértices y de aristas, ver figura 6.

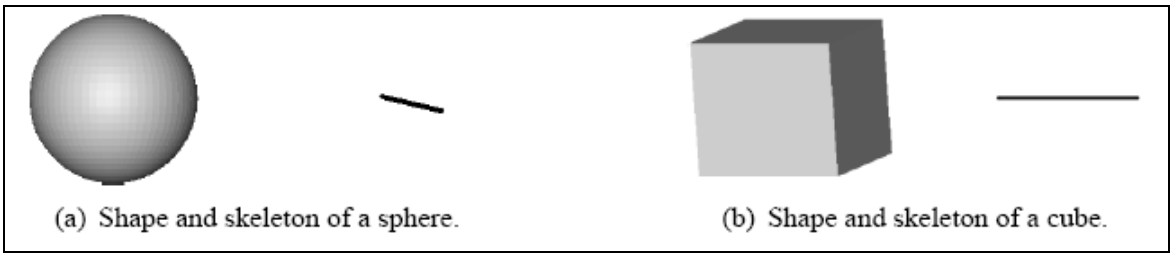


Figura 6. Comparando forma y esqueleto de una esfera con los de un cubo. Nótese que la eliminación de vértices y artistas hace que pierda detalle Imágenes extraídas de [Brennecke04].

En [Cyr01] se presentó un método en que se calculan *n vistas* alrededor de cada objeto 3D, dichas vistas comprenden un conjunto de *imágenes* de las cuales se determinan las que *caracterizan* al objeto, llamados *aspectos*, ver figura 7.

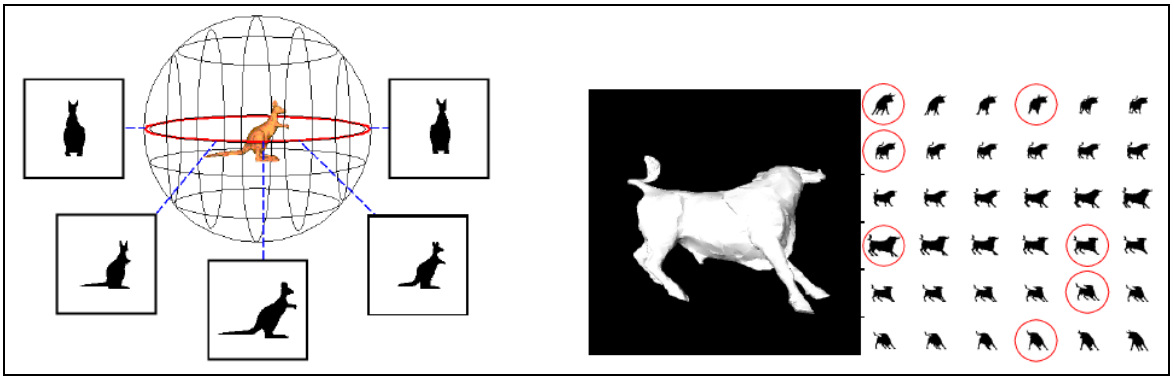


Figura 7. Reconocimiento de objetos 3D usando grafos de aspectos. A la izquierda se ilustra un ejemplo en que se generan *aspectos* de un objeto 3D. Los aspectos representativos se conservan (lado derecho de la figura). Nótese que sólo *aspectos* alrededor del objeto son considerados.

En [Osada01] se utilizan distribuciones de forma (*shape dsitribution*), dichas distribuciones (secuencias), son posteriormente comparadas por medio de métricas de Minkowski. Las distribuciones de forma son múltiples, el ángulo entre tres puntos aleatorios dentro de la superficie del modelo, la distancia entre dos puntos aleatorios en la superficie del modelo, entre otras.

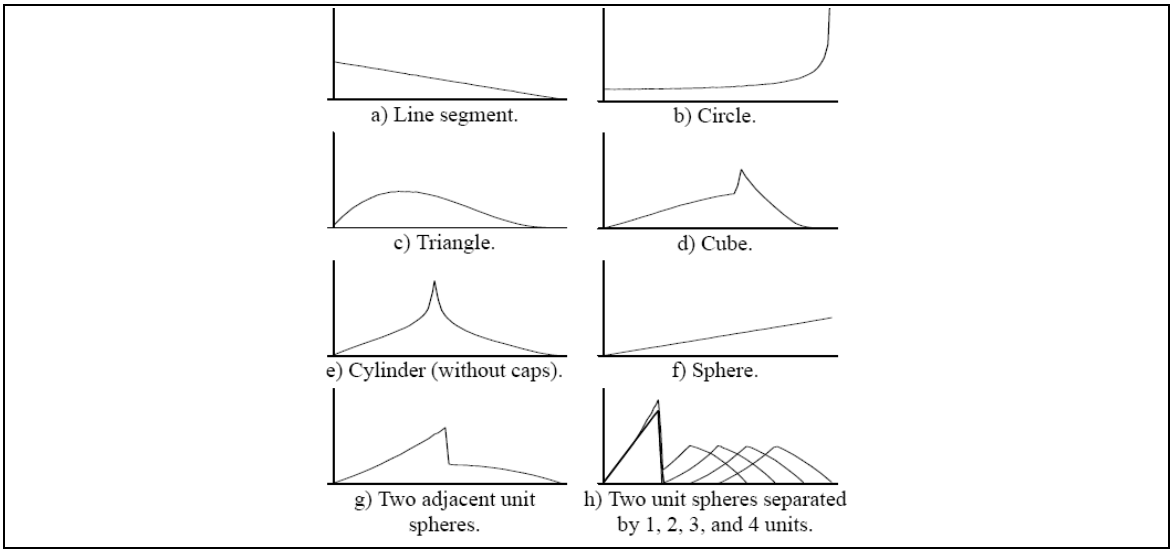


Figura 8. Emparejando modelos 3D con distribuciones de forma. La idea planteada en [Osada01] es la caracterizar los modelos 3D por medio de distribuciones de forma. Nótese que es casi arbitraria la determinación de las distribuciones de forma.

En 2003 se presentó en [Hlavatý03] un estudio de los métodos para *extracción de características* de modelos 3D. Este estudio hace énfasis en los métodos en los que alguna etapa de extracción / selección / construcción de características es inminente, los autores consideran que la parte más importante de un método de búsqueda de similitud es la etapa de obtención de un vector de características:

> *"Exactly, the main attention is pointed to the way of feature extraction, i.e., a shape description of 3D object by a feature vector that consists of real digits. We can say that this is the most important part of the whole system. The shape of the 3D object is described by feature vector that serves as a search key in the database. If an unsuitable feature extraction method had been used, the whole retrieval system would not be usable"*

La hipótesis de este trabajo señala que no necesariamente todo método de similitud entre figuras (y quizá otras entidades), requiere de dicha etapa.

Hlavatý et al finalmente postularon que los métodos para extracción de características de modelos 3D se clasifican en:

1. *Métodos basados en vistas* 2D. El enfoque de éstos es representar los objetos 3D como una serie de imágenes 2D. Por ejemplo, Grafo de Aspectos [Cyr01].

2. *Métodos basados en histogramas*. Estos métodos se fundamentan en ideas matemáticas que puedan se usadas para describir un objeto 3D. Por

ejemplo, Análisis de Componentes Principales [Tangelder03], matemáticas diferenciales [Boyer02] [Mokhtarian01] [Quek01] [Yarger00], armónicas esféricas [Funkhouser03], entre otras.

3. *Métodos basados en topologías*. Estos métodos buscan representar la topología de objeto por medio de grafos, y así comparar dichos grafos, esto comúnmente es computacionalmente costoso, estos métodos son altamente sensibles al ruido. Por ejemplo, grafos Reeb [Hilaga01]

4. *Métodos basados en el error*. Estos métodos tienen que resolver dos problemas: 1) estimar la posición adecuada del objeto 3D y 2) calcular un error (en volumen) entre las superficies de los objetos a comparar. Un ejemplo de estos métodos se encuentra en [Novotni01].

2.1 CÓMPUTO DE SIMILITUD ENTRE FIGURAS USANDO ART

Los trabajos que hasta ahora involucran algún tipo de reconocimiento o comparación entre objetos 3D / figuras aplicando de alguna forma *Redes Neuronales Artificiales ART* (Adaptive Resonante Theory) son métodos para reconocer objetos 3D a partir de vistas 2D (imágenes), ejemplos de estos métodos se encuentra en [Ming], [Chung] [Bradski95], y [Seibert92].

A diferencia de los métodos anteriormente mencionados, no existe registro de alguna forma de hacer similitud entre figuras directamente (sin construcción de vistas 2D) usando alguna RNA ART (ART-1, ART-2, Fuzzy-ART, ART-MAP, etc.). Propuestas de métodos de similitud entre imágenes de figuras se derivan principalmente de [Nazuno91] y [Sossa98].

2.2 CÓMPUTO DE SIMILITUD ENTRE FIGURAS USANDO DTW

Aunque *Dynamic Time Warping* y sus variantes (*Fast Dynamic Time Warping*) ha sido utilizado ampliamente para la comparación de secuencias / arreglos / series de tiempo / vectores / señales, no se ha reportado trabajo conocido por los autores para la comparación de objetos 3D.

REFERENCIAS DEL CAPÍTULO

[Guzmán68] Guzmán A. “Computer Recognition of Three-dimensional Objects in a Visual Scene”. Ph. D. Thesis, Electrical Engineering Department, M. I. T., December 1968. Also available as Project MAC Technical Report MAC TR 59. AD-692-200.

[Roberts65] Roberts L.G. “Machine perception of three-dimensional solids”. *Optical an Electrooptical information processing*, pp. 159-197, J T Tippett et al (eds) MIT Press 1965.

[Shilane04] Shilane P., Kazhdan M., Min P., Funkhouser T. “The Princeton Shape Benchmark”. *In Proceedings of Shape Modeling International*, 2004.

[Brennecke04] Brennecke A., Isenberg T. “3D Shape Matching Using Skeleton Graphs”. *In Proceedings of Simulation und Visualisierung*, pp. 299-310, 2004

[Cyr01] Cyr C. M., Kimia B. “3D Object Recognition Using Shape Similarity-Based Aspect Graph”. *In Eighth International Conference on Computer Vision* (ICCV-01), pp. 254-261, 2001.

[Osada01] Osada R., Funkhouser T., Chazelle B., Dobkin D. “Matching 3D models with shape distributions”. pp. 154-166, 2001.

[Hlavatý03] Hlavatý T., Skala V. “A Survey of Methods for 3D Model Feature Extraction”. *Bulletin of IV. Seminar Geometry and Graphics in Teaching Contemporary Engineer*, No: 13/03, pp. 5-8, 2003.

[Boyer02] Boyer K. L., Srikantiah R., Flynn P. J. “Salience Sequential Surface Organization for Free-Form Object Recognition”. *Computer Vision and Image Understanding*, Vol. 88, No. 3, 2002.

[Mokhtarian01] Mokhtarian F., Khalili N., Yuen P. “Multi-scale free-form 3D object recognition using 3D models”. *Image and Vision Computing*, Vol. 19, pp. 271-281, 2001.

[Quek01]Quek K. H., Yarger R. W. I., Kirbas C. “Surface Parameterization in Volumetric Images for Curvature-based Feature Classification”. *IEEE Transactions on Systems*, Man and Cybernetics. 2001.

[Yarger00]Yarger R. W. I., Quek K. H. “Surface Parameterization in Volumetric Images for Feature Classification”. *IEEE Inter. Symposium of Bio-Eng.* BIBE 2000, pp. 297-303, 2000.

[Funkhouser03] Funkhouser T., Min P., Kazhdan M., Chen J., Halderman A., Dobkin D. “A Search Engine for 3D Models”. *ACM Transactions on Graphics*, Vol. 22, Issue 1, pp. 83-105, 2003.

[Tangelder03] Tangelder J. W. H., Veltkamp R. C. “Polyhedral Model Retrieval Using Weighted Point Sets”. *International Journal of Image and Graphics*, Vol. 3, No. 1, pp. 209-229, 2003.

[Hilaga01] Hilaga M., Shinagawa Y., Kohmura T., Kunii T. L. “Topology Matching for Fully Automatic Similarity Estimation of 3D shapes”. *SIGGRAPH 2001*, pp. 203-212, 2001.

[Novotni01] Novotni M., Klein R. “A Geometric Approach to 3D Object Comparison”. *Int. Conf. on Shape Modeling and Applications*, pp. 154-166, 2001.

[Bradski95] Bradski G., Grossberg S. “Fast Learning VIEWNET Architectures for Recognizing 3-D Objects from Multiple 2-D Views”. *Neural Networks Special Issue on ATR*. pp. 1053-1080, 1995.

[Seibert92] Seibert M., Waxman A. M. “Adaptive 3-D Object Recognition from Multple Views”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 14, No. 2, pp. 107-124. 1992.

[Nazuno01] Figueroa-Nazuno, Kashiwamoto-Yabuta J., Vargas-Medina E. “Clasificación y representación de resultados experimentales en 3-D, mediante una red neuronal*”. XXXIV Congreso Nacional de Física*, 1991.

[Ming] Ming-Ching L., Chong-Juch L., Hon-Son D. “A Neural Network Approach to 3-D object identification and pose estimation”. *IEEE International Joint Conference on Neural Networks*. pp. 2600-2605.

[Chung] Chung L. W., Yuan L. F., Kuo T. C., Lingutla T. “A Hierarchical Multiple View Approach to Three Dimensional Object Recognition”. *IEEE Transactions on Neural Networks*, vol. 2, no. 5, pp. 568-576.

[Sossa98] Sossa H., Rayon P. Figueroa J. “2D Object Recognition by Indexing through a modified Art-2 Neural Network”. *The Fourth World Congress on Expert Systems,* 1998.

3 ESTRUCTURA CONCEPTUAL DEL PROBLEMA DE SIMILITUD

De acuerdo a la *Real Academia de la Lengua Española*, similitud se refiere a la *cualidad de ser semejante*. Dicha expresión recursiva nos deja aún con la interrogante de qué es la similitud. Este capítulo tiene por objetivo *esbozar* una *estructura conceptual* el problema de similitud.

Aunque no existen marcos conceptuales para el problema de similitud, de forma tácita se ha propuesto el siguiente paradigma para calcular similitud entre entidades [Hlavatý03]:

1) Definir *principales características* de las *entidades* a comparar.
2) Definir una *función de similitud*, basado en dichas características.
3) *Calcular* similitud entre las entidades de acuerdo a 1) y 2).

A continuación en la figura 9 se ilustra el paradigma anterior.

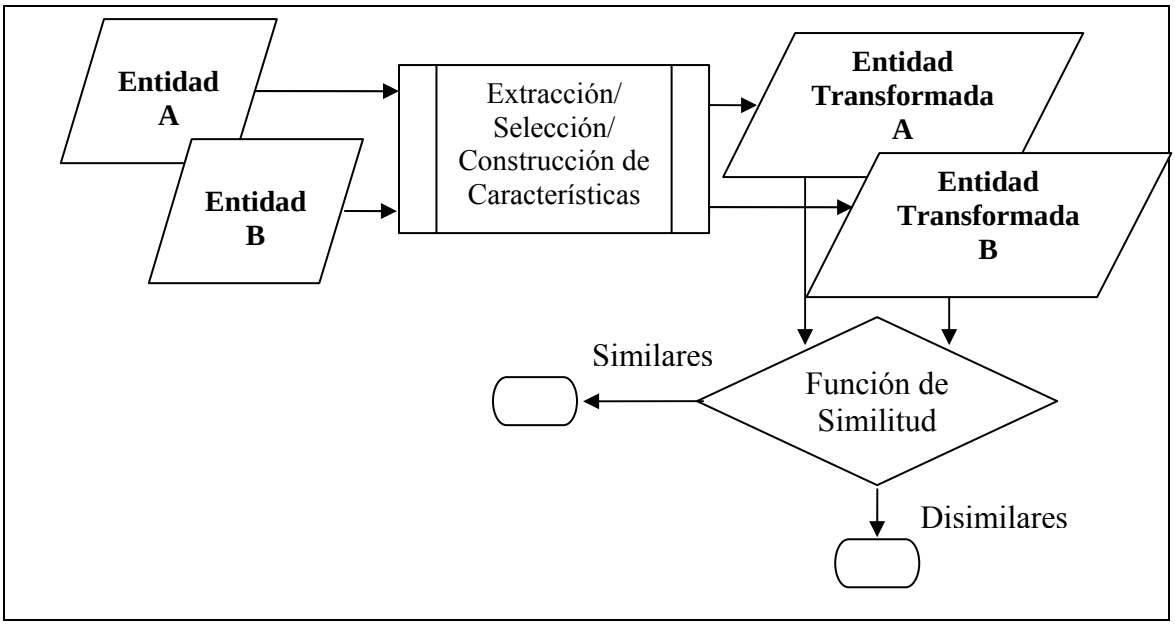


Figura 9. El *paradigma* clásico para el problema de calcular similitud entre entidades. La *función de similitud* se ha ilustrado como un proceso de toma de decisión binario. Otros casos pueden mapearse al paradigma.

El lector puede reconocer la importancia de dicho paradigma en áreas como reconocimiento de voz, reconocimiento de patrones, entre otras.

El primer paso del paradigma asegura la necesidad de seleccionar/extraer/construir características de cada entidad a comparar. Dicho enfoque requiere conocer a priori, lo *significativo* de la entidad en cuestión, lo restante por lo tanto, no es información necesaria. Otra justificación, es la minimización del costo computacional de la función de similitud, al considerar menos información de cada entidad para el proceso de comparación.

El segundo paso, acota el paradigma a ciertas entidades (las que pueden ser comparadas con una función de similitud), no hace explícita la existencia de diferentes teorías de la medida [Krantz71] [Luce90] [Suppes89], que aún siguen siendo exploradas activamente.

Para realizar alguna estructura conceptual del problema de similitud, debemos plantearnos la siguiente pregunta:

¿Cuáles son los elementos más *importantes* en el problema de similitud?

De forma general, la estructura conceptual del problema de similitud consiste de tres elementos fundamentales, uno de ellos con carácter de opcional. En la figura 10 se ilustra lo anterior.

- ✓ Entidades a comparar.
- ✓ Criterio de comparación.
  - o Descriptor de las propiedades que interesen.
  - o Comparación de las entidades como un todo

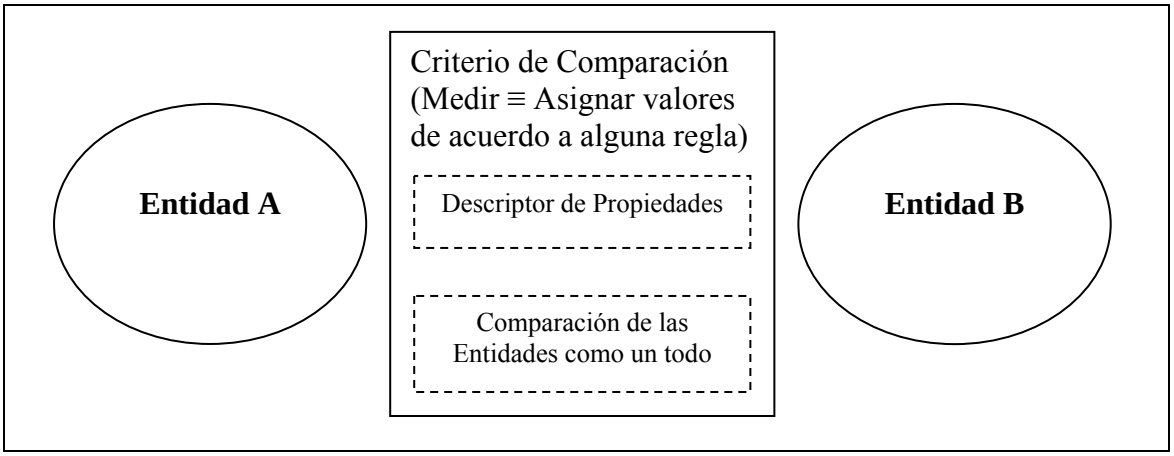


Figura 10. Estructura conceptual del problema de similitud. Entidades a comparar (cuantificables), criterio de comparación (proceso para asignar números en base a alguna regla).

En la figura 10, se muestra la estructura conceptual propuesta en este trabajo. La línea punteada denota elementos opcionales dentro de la estructura conceptual. En las siguientes subsecciones se justifican cada uno de estos elementos.

3.1 ENTIDADES A COMPARAR

El problema de similitud es libre de contexto, esto es, las técnicas para calcular similitud (a excepción de auto-similitud), requieren de al menos dos entidades, estas entidades pueden ser secuencias, matrices, grafos, etc.

Las entidades a comparar sólo deben ser *cuantificables*. He aquí el primer concepto teórico sobre los límites del problema de similitud. De acuerdo a [Krantz71], [Luce90], y [Suppes89], medir es *"el acto o proceso de asignar números al fenómeno de acuerdo a alguna regla"*, todo aquél fenómeno que no pueda ser medido, no puede ser comparado. La teoría de la medida sigue siendo explorada activamente. Sin embargo, se han propuesto diferentes sistemas de medida, que abren la posibilidad de que las funciones de similitud puedan ser un grado de similitud, una razón de similitud, un rango de similitud, etc., y no simplemente un mapeo uno a uno de un dominio y un contradominio.

3.2 CRITERIO DE COMPARACIÓN

El criterio de comparación, es el proceso o acto de medir, el lector podrá notar la gran relación entre las entidades a comparar y el criterio de comparación. Ello indica la dependencia del criterio de comparación con el conjunto de entidades a comparar.

La estructura conceptual propuesta en este trabajo, nos indica que el primer límite para comparar dos entidades es que éstas deben ser cuantificables.

De lo anterior, se expresa que la función de similitud del paradigma clásico, es un subconjunto de criterios de comparación, ya que el criterio de comparación, como proceso o acto de asignar números de acuerdo a una regla, contempla otras formas de medir la similitud entre entidades distintas.

Un aspecto importante del problema de calcular similitud, es la *validación* de las clases o grupos que se puedan generar de una técnica dada. No existe una clasificación correcta de las cosas per se.

La comparación de entidades no se puede racionalizar como un proceso en el que se establece una relación de orden binaria. Por ejemplo, para cualquier elemento de los números naturales positivos, podemos determinar los *n* elementos más similares sin

ninguna dificultad. Esto lo podemos realizar ya que los números naturales positivos, ya tienen un orden preestablecido, y por lo tanto el criterio de comparación puede ser una relación de orden binaria. En este caso, se puede afirmar si un número es más que otro, sin embargo, existen casos en que las entidades a comparar no tienen dicho orden preestablecido, no hay imágenes más que otras, o señales más que otras, etc.

3.2.1 DESCRIPTOR DE PROPIEDADES

El descriptor de propiedades tiene por objetivo representar las entidades a comparar de forma más conveniente al criterio de comparación.

La existencia de más de un descriptor en el problema de similitud, concerniente a un contexto en particular; nos indica un asunto de fondo muy interesante. De forma similar al criterio de comparación, no existe un descriptor universal, esto es, un descriptor que nos de la única información importante para cualquier entidad a comparar, el ejemplo más reciente puede ser encontrado en [Shilane04].

De lo anterior, se puede afirmar que la definición de los descriptores de propiedades no es independiente ni de las entidades a comparar, ni del criterio de comparación. Y por lo tanto no existen pasos para abordar un problema de similitud, sino una estructura conceptual, que ayuda a entender los elementos del problema y su relación entre sí.

3.2.2 COMPARACIÓN DE LAS ENTIDADES COMO UN TODO

Un criterio de comparación alternativo a la descripción de propiedades es la comparación de las entidades completas. Las técnicas que utilizan dicho criterio se pueden denominar técnicas "*Gestalt*", y entre las más representativas se encuentra el Análisis de Componentes Principales (Principal Component Analysis, PCA).

La importancia de calcular similitud entre entidades ha atraído el interés de muchos investigadores no por ser un problema abierto, sino que porque puede ser el camino para resolver otros problemas, como el predicción de series de tiempo, reconocimiento de patrones, descubrimiento de reglas, etc.

REFERENCIAS DEL CAPÍTULO

[Krantz71] Krantz, D. H., Luce, R. D., Suppes, P., & Tversky, A. (1971). *Foundations of measurement: Vol. 1. Additive and polynomial representations*. New York: Academic.

[Luce90] Luce, R. D., Krantz, D. H., Suppes, P., & Tversky, A. (1990). *Foundations of measurement: Vol. 3. Representation, axiomatization, and invariance*. San Diego, CA: Academic.

[Suppes89] Suppes, P., Krantz, D. H., Luce, R. D., & Tversky, A. (1989). *Foundations of measurement: Vol. 2. Geometrical, threshold, and probabilistic representations*. San Diego, CA: Academic.

[Shilane04] Shilane P., Kazhdan M., Min P., Funkhouser T. "The Princeton Shape Benchmark". *In Proceedings of Shape Modeling International*, 2004.

[Hlavatý03] Hlavatý T., Skala V. "A Survey of Methods for 3D Model Feature Extraction". *Bulletin of IV. Seminar Geometry and Graphics in Teaching Contemporary Engineer*, No: 13/03, pp. 5-8, 2003.

4 RNA FUZZY ART

Una característica de la memoria humana es su habilidad para aprender cosas nuevas sin necesariamente olvidar las ya aprendidas. Esta habilidad ha sido descrita como el dilema de *estabilidad/plasticidad* [Carpenter87a]. La Teoría de Resonancia Adaptativa ART (*Adaptative Resonance Theory*) fue propuesta como una teoría de procesamiento de información cognoscitiva humana para la solución del dilema de *estabilidad/plasticidad* [Grossberg76a] [Grossberg78b]. Un sistema de memoria debe ser un sistema adaptativo a entradas *significantes* y que a la vez permanezca estable en respuesta a entradas *irrelevantes*, ello implica un mecanismo para diferenciarlas, adicionalmente, retener la información aprendida mientras se clasifican nuevos patrones.

La teoría ART condujo al desarrollo de una serie de modelos de RNA de tiempo-real para el aprendizaje no supervisado de categorías y reconocimiento de patrones (en forma de secuencias/series de tiempo/arreglos/vectores). *Adaptative* –adaptativo- proviene de su condición *estable/plástica*, y *Resonance* –resonancia- al mecanismo de conmutación entre estados. En ART un estado de resonancia puede alcanzarse de dos formas: i) cuando la RNA ha aprendido anteriormente a reconocer un patrón de entrada *a*, y éste vuelve a ser presentado en los *n* momentos posteriores (durante la resonancia, un proceso adaptativo reforzará la memoria del patrón almacenado) y ii) cuando el vector de entrada no es inmediatamente reconocido y la RNA busca a través de los patrones almacenados un patrón semejante; si no se encuentra dicho patrón, la RNA entrará en un estado de resonancia donde el nuevo patrón será almacenado por primera vez. Es de esta forma que las RNA ART permite el reconocimiento y clasificación de patrones de entrada de acuerdo a su *similitud*.

Las RNA ART son redes neuronales auto-organizadas que pueden ser entrenadas de forma supervisada o no supervisada. Por ejemplo la RNA ART1 es utilizada para clasificar patrones de entrada binarios, las RNA ART2 y ART3 son usadas para clasificar patrones análogos y las Fuzzy ART aplicadas a sistemas definidos con información difusa, cuyos modelos de aprendizaje pertenecen al grupo de algoritmos no supervisados. La familia de las redes ART que se entrenan de manera supervisada son designadas como redes ARTMAP.

Las secuencias/series de tiempo/arreglos/vectores de entrada, deben ser normalizadas, es decir, los elementos deben estar entre 0 y 1, y de la misma longitud. Este último criterio suele ser una gran desventaja cuando se quiere categorizar *patrones* de diferente tamaño.

Es común en la literatura encontrar las arquitecturas ART, comprendidas por *capas de actividad*, *capas de contención*, *memoria de corto* y *largo plazo*, metáforas de neuronas, sinápsis, etc., sin embargo, la RNA Fuzzy ART (y otras ART), pueden ser estudiadas como paradigmas, algoritmos, y heurísticas.

El lector puede corroborar que el algoritmo presentado más adelante es claro y corresponde a la idea presentada por Carpenter et. al en 1991.

A continuación se muestra el algoritmo de la RNA Fuzzy ART.

Algoritmo RNA Fuzzy ART

1. Establecer parámetros: $N \in \aleph^*$, $M \in \aleph^*$, $\alpha > 0$, $\beta \in [0,1]$, y $\rho \in [0,1]$
2. Crear matriz de pesos: $w_{1,1} = \ldots = w_{N,2M} = 1$
3. Mientras existan vectores de entrada, hacer
   3.1. Establecer el vector/secuencia/serie de tiempo de entrada $Q_i = \langle q_1, q_2, \ldots, q_M \rangle$
   3.2. Calcular el patrón de entrada $I = (q_1, q_2, \ldots, q_M, q_1^c, q_2^c, \ldots, q_M^c)$, donde $q_j^c = 1 - q_j$
   3.3. Mientras **no** sea encontrada la categoría del patrón de entrada, hacer
      3.3.1. De 1 a *N*, hacer
      $T_j(I) = \left| I \wedge w_{j,*} \right| \Big/ \alpha + \left| w_{j,*} \right|$, donde $(I \wedge w_{j,*})_i = \min(q_i, w_{j,i})$ y $|x| = \sum_1^{2M} x_i$
      Fin de hacer
      3.3.2. Determinar la posible categoría
      $T_J = \max\{T_j : j = 1, \ldots N\}$
      3.3.3. Si $\left| I \wedge w_J \right| \Big/ |I| \geq \rho$, entonces *J* es la categoría del patrón de entrada y actualizar matriz de pesos $w_{J,*}^{new} = \beta \times (I \wedge w_{J,*}^{old}) + (1 - \beta) \times w_{J,*}^{old}$
      3.3.4. De lo contrario, deshabilitar la categoría *J*, e ir al paso 3.3
   Fin de hacer
Fin de hacer

A continuación se aborda cada paso del algoritmo Fuzzy ART.

Paso 1. El algoritmo Fuzzy ART requiere de 5 parámetros. *N* es el número de categorías o clases, este parámetro tiene que ser un valor entero positivo, usualmente mayor a 1; *M* es el número de elementos o longitud en las secuencias/series de tiempo/arreglos/vectores de entrada, α (alfa) es un valor real mayor que 0, β (beta) es un valor real entre 0 y 1, y ρ (rho) también es un valor real entre 0 y 1, éste también es conocido como el parámetro de vigilancia. El parámetro ρ es un umbral de similitud, si este valor es igual a 0, sólo se creará una categoría, en el otro extremo, si este valor es igual a 1, sea creará una categoría por cada secuencia/serie de tiempo/arreglo/vector distinto.

Paso 2. La matriz de pesos es la "memoria" del sistema de aprendizaje, esta matriz se denomina memoria a largo plazo o matriz de pesos adaptativos; la inicialización de la matriz a 1 permite categorizar los patrones de entrada de forma más expedita; las dimensiones de la matriz es de *N*, 2*M*; *N* es el número de columnas y *2M* el número de filas, el número de filas debe estar de acuerdo al esquema de codificación de los secuencias/series de tiempo/arreglos/vectores de entrada. La codificación es utilizada en este algoritmo para evitar en cierto grado la proliferación de categorías.

Paso 3. Los pasos contenidos en este paso se repetirán hasta que *todas* las secuencias /series de tiempo/arreglos/vectores de entrada sean categorizadas o clasificadas.

Pasos 3.1 y 3.2. Estos pasos establecen la diferencia entre las secuencias/series de tiempo/arreglos/vectores *Q* y su codificación en el *patrón* de entrada *I*, como se mencionó anteriormente el costo de duplicar la longitud de los arreglos de entrada se compensa con la posibilidad de reducir la proliferación de categorías, dicha codificación fue metafóricamente aplicada de las células ON/OFF.

Paso 3.3. Es ciclo terminará cuando el patrón de entrada sea categorizado o clasificado.

Paso 3.3.1. El propósito de este paso es calcular por cada columna de la matriz, es decir, por cada patrón "prototipo", un patrón *T*, donde cada elemento de este patrón es un cociente del patrón de entrada, la matriz de peso y el valor de α.

Paso 3.3.2. Una vez que se ha calculado los patrones *T* (*N* patrones), se determina el patrón máximo, es decir, el patrón *T* cuya sumatoria de elementos sea mayor que el resto de los patrones *T*. En esta etapa se dice que existe metafóricamente una capa de contención.

Paso 3.3.3. El índice del patrón *T*, indica la columna donde se encuentra el patrón prototipo que posiblemente sea similar al patrón de entrada *I*, es aquí donde el

parámetro de vigilancia ρ cobra su importancia. Si el patrón de entrada *I* es semejante al patrón almacenado en la columna *J*, entonces se redefine el patrón almacenado en la matriz de pesos.

Paso 3.3.4. Si el vector de entrada I, no es semejante al patrón almacenado en la columna *J*, se requiere descartar dicha columna y regresar al paso 3.3. El lector puede detectar que el parámetro de vigilancia ρ es decisivo no solo como criterio de similitud, también puede determinar en gran parte el número de clases o categorías creadas. De forma similar, también puede restringir la posibilidad de encontrar alguna categoría a los patrones de entrada.

4.1 CASO SET/SUPERSET

En esta subsección se ilustra el caso set/superset binario y análogo. En [Carpenter87a] se hace particular énfasis en el caso set/superset; este ejemplo es importante porque ilustra la capacidad de la RNA Fuzzy ART para abordar el dilema plasticidad/estabilidad. El primer caso de set/superset abordado es cuando se tratan de patrones de entrada binario, el secundo caso es cuando los patrones de entrada son análogos. En la figura 11 se muestran tres patrones binarios; el patrón 2 es un súper-conjunto (superset) del patrón 3 (set), ya que coinciden en el elemento número 5, el patrón 1 es un patrón de control.

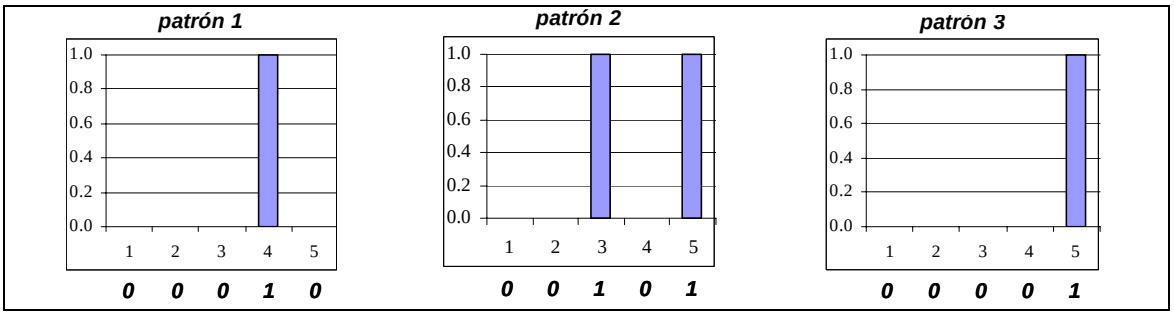


Figura 11. Tres patrones binarios para ejemplificar el caso *set/superset* en una RNA Fuzzy ART.

El siguiente código en lenguaje C++ ejemplifica el dilema estabilidad/plasticidad, el ejercicio consta de presentar tres diferentes patrones de forma secuencial, el primer patrón (de control), será disímil a todo lo existente en la matriz de "memoria" de la RNA Fuzzy ART, y por lo tanto será categorizado dentro la clase 0.

El propósito es mostrar cómo el patrón 3 y 2 (set/superset) son categorizados dentro de la misma clase. Los módulos `FuzzyArt.h` y `FuzzyArt.cpp` son integrados en el Anexo A.

Objetivo: Ejemplo clásico para ilustrar el dilema plasticidad/ Estabilidad en patrones de entrada binarios
Salida: Cómputo de Similitud y Clasificación de "patrones"

```
#include "FuzzyArt.h"
#include <iostream.h>

void main() {
  double input[5][5] = {{0, 0, 0, 1, 0},     // patron 1
                        {0, 0, 1, 0, 1},     // patron 2
                        {0, 0, 0, 0, 1},     // patron 3
                        {0, 0, 1, 0, 1},     // otra vez patron 2
                        {0, 0, 0, 0, 1}};    // otra vez patron 3
  CFuzzyART net;
  net.InitializeFuzzyART(0.95, 0.5, 0.01); //Rho, Alpha and Beta

  for (int i=0; i<5; i++) {
    net.Input(input[i]);
    net.Step2();
    cout<< "El patron de entrada pertenece a la clase" <<net.J
                                                       <<endl;
  }
}
```

La salida correspondiente al código anterior es:

```
El patron de entrada pertenece a la clase 0
El patron de entrada pertenece a la clase 1
El patron de entrada pertenece a la clase 1
El patron de entrada pertenece a la clase 1
El patron de entrada pertenece a la clase 1
Press any key to continue_
```

Figura 12. Clasificación de patrones de entrada binarios (plasticidad/estabilidad).

Los patrones de prueba son almacenados en la matriz `input`, por simplicidad el patrón 2 y 3 son repetidos. La intención es presentar el patrón 2 y una vez categorizado, presentar el patrón 3, este último deberá ser categorizado en la misma clase del patrón 2, ya que coinciden el quinto elemento de la secuencia, es decir el patrón 3 es similar al patrón 2 en el quinto elemento.

Una vez que se da la plasticidad, es momento de probar la estabilidad presentando nuevamente el patrón 2 y posteriormente el patrón 3.

`CFuzzyART` es la clase designada para crear instancias del algoritmo de la RNA Fuzzy ART; el método `InitializeFuzzyART`, asigna los valores de los parámetros ρ (rho), α (alfa), y β (beta), el número de categorías o clases N y la longitud de los patrones de entrada M, son definidos en el archivo `FuzzyART.h`.

Por último, el ciclo presentará los patrones de entrada (`Input();`), que equivale a los pasos *3.1* y *3.2* del algoritmo de la RNA Fuzzy ART, presentado anteriormente. De forma análoga, la categorización (`Step2();`), correspondiente al paso *3.3*.

El índice de la clase indica la pertenencia de un patrón a una clase, ello no implica que los patrones prototipo dentro de la matriz (columnas) tengan un orden específico, ni mucho menos que se pueda establecer una relación binaria entre dichos patrones.

```
Objetivo: Ejemplo clásico para ilustrar el dilema plasticidad/
Estabilidad en patrones de entrada análogos
Salida: Cómputo de Similitud y Clasificación de "patrones"
```

```
#include "FuzzyArt.h"
#include <iostream.h>

void main() {
  double input[5][5] = {{0.2, 0.7, 0.1, 0.5, 0.4},      // patron 1
                        {0.1, 0.35, 0.050, 0.250, 0.2},  // patron 2
                        {0.4, 1.4, 0.2, 1.0, 0.8},       // patron 3
                        {0.8, 1.0, 0.2, 1.4, 0.4},       // patron 4
                        {0.4, 1.0, 0.8, 0.4, 1.0}};      // patron 5
  CFuzzyART net;
  net.InitializeFuzzyART(0.85, 0.5, 0.5);//Rho, Alpha and Beta

  for (int i=0; i<5; i++) {
    net.Input(input[i]);
    net.Step2();
    cout<<"El patron "<<i+1<<" pertenece a la clase " <<net.J<<endl;
    }
  }
```

La salida correspondiente al código anterior es:

```
El patron 1 pertenece a la clase 0
El patron 2 pertenece a la clase 0
El patron 3 pertenece a la clase 1
El patron 4 pertenece a la clase 1
El patron 5 pertenece a la clase 2
Press any key to continue
```

Figura 13. Clasificación de patrones de entrada análogos (plasticidad/estabilidad).

Conforme se realizan pruebas experimentales con Fuzzy ART se puede generalizar que el valor del parámetro de vigilancia ρ (rho), oscila entre 0.7 y 0.96, dicho valor depende de los patrones de entrada.

El programa anterior ejemplifica también el caso de *set/superset* en patrones de entrada análogo.

Aunque los autores de la RNA Fuzzy ART mencionan que la capacidad de categorizar patrones análogos de la arquitectura se debe a la sustitución del operador *min* de la teoría de conjuntos difusos, la RNA Fuzzy ART comprende varias modificaciones sustanciales.

Un caso reportado en [Angeles04a] es el de inversión de patrones/vectores, cosa útil para la clasificación de secuencias/series de tiempo/etc.

A continuación, en la figura 14 se muestra la clasificación creada por RNA Fuzzy ART para los 5 patrones de entrada, del segundo programa de este capítulo.

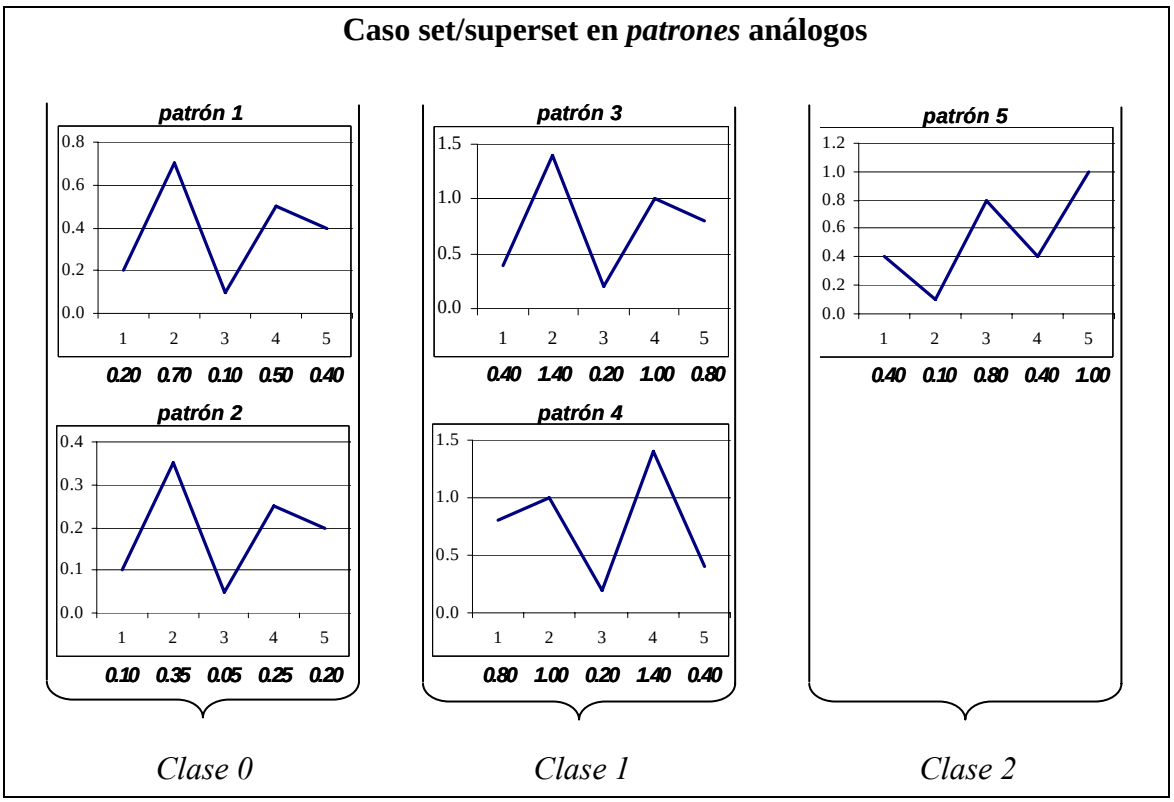


Figura 14. Búsqueda de similitud y clasificación de patrones de entrada análogos. Caso *set/superset* de patrones análogos.

Otro ejemplo puede ser encontrado en [Angeles04b], donde se calcula similitud y clasifica señales sísmicas, en la figura 15 se muestra lo anterior.

Nótese que la clasificación no tiene implicaciones sobre la naturaleza de los sismos, en [Angeles05c] puede ser encontrada una clasificación de zonas sísmicas a través de espectros de respuesta.

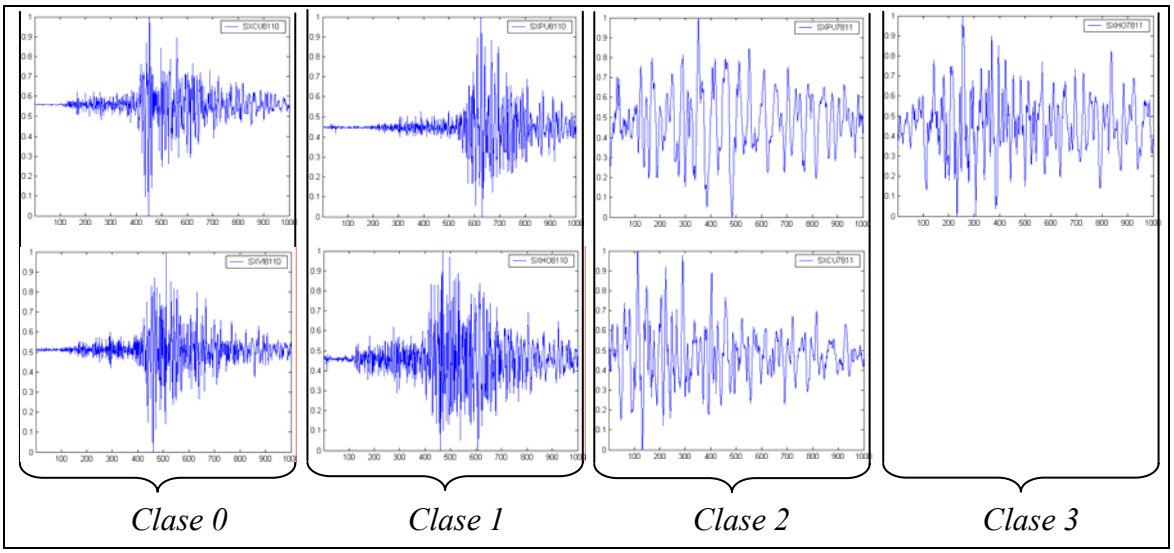


Figura 15. Búsqueda de similitud y clasificación de señales sísmicas.

4.2 RNA's ART CLÁSICAS

ART se ha conceptualizado como teoría, dicho enfoque está basado en algunos aspectos matemáticos y neurofisiológicos, principalmente Grossberg y Carpenter ha desarrollado variantes de su primer trabajo sobre la arquitectura ART, algunas de estas variantes son:

- **ART-1**. Clasifica únicamente secuencias binarias [Carpenter87a].
- **ART-2**. Similar a ART-1, clasifica secuencias binarias y de valores reales [Carpenter87b].
- **ART-2A**. Versión acelerada de ART-2 [Carpenter91a].
- **ART-3**. Esta arquitectura es una extensión a ART que incorpora una metáfora de *transmisores químicos* para controlar el proceso de búsqueda en una estructura ART jerárquica [Carpenter90].
- **ARTMAP**. Versión supervisada de ART, puede aprender mapeos arbitrarios de patrones binarios [Carpenter91b].
- **Fuzzy ART**. Esta red es una síntesis entre ART y lógica borrosa [Carpenter91c].
- **Fuzzy ARTMAP**. Versión supervisada de Fuzzy ART [Carpenter92].

Para el lector interesado en *RNA's ART* y sus variantes se recomienda el trabajo [Heins95].

REFERENCIAS DEL CAPÍTULO

[Angeles04a] Angeles-Yreta A., Solís-Estrella H., Landassuri-Moreno V., Figueroa-Nazuno J. “Similarity Search In Seismological Signals”. *Fifth Mexican Internacional Conference on Computer Science*. pp. 50-56. 2004.

[Angeles04b] Angeles-Yreta A, Figueroa-Nazuno J. “Análisis de Similitud en Señales Sísmicas con Red Neuronal Artificial Fuzzy ART”. *Congreso Nacional de Física 2004*. ISSN 0187-4713, pp. 114, 2004.

[Angeles05c] 4. Angeles-Yreta A. et al. “Clasificación y Similitud de Espectros de Respuesta de Aceleración”. *XV Congreso Nacional de Ingeniería Sísmica*. Artículo I-20. 2005.

[Heins95] Lucien G. Heins and Daniel R. Tauritz. Adaptive resonance theory (art): An introduction. Technical Report 95-35, Leiden University, 1995.

[Grossberg76a] Grossberg S. “Adaptive pattern classification and universal recoding. I. Parallel development and coding of neural feature detectors”. Biol. Cybernet 23, pp. 121-134, 1976.

[Grossberg76b] Grossberg S. “Adaptive pattern classification and universal recoding. II. Feedback, expectation, olfacation, and illusions”. Biol. Cybernet. 23, pp.187-202, 1976.

[Carpenter87a] Carpenter G., Grossberg S. “A Massively Parallel Architecture for a Self Organizing Neural Pattern Recognition Machine”. *Computer Vision, Graphics, and Image Processing*, Vol. 37, pp.54-115, 1987.

[Carpenter87b] Carpenter G., Grossberg S. “ART 2: Self-organization of stable category recognition codes for analog input patterns”. *Applied Optics* 26(23): pp. 4919-4930, 1987.

[Carpenter91a] Carpenter G., Grossberg S., Rosen D.B. “ART2-A: An Adaptive Resonance Algorithm for Rapid Category Learning and Recognition”. *Neural Networks*, Vol. 4, pp.493-504, 1991.

[Carpenter91b] Carpenter G., Grossberg S., Reynolds J. H. “ARTMAP: Supervised Real-Time Learning and Classification of Nonstationary Data by a Self-Organizing Neural Network”. *Neural Networks*, Vol. 4, pp.565-588, 1991.

[Carpenter91c] Carpenter G., Grossberg S. “Fuzzy ART: Fast Stable Learning and Categorization of Analog Patterns by an Adaptive Resonance System”. *Neural Networks*, Vol. 4, pp.759-771, 1991.

[Carpenter90] Carpenter G., Grossberg S. “ART3: Hierarchical Search Using Chemical

Transmitters in Self-Organizing Pattern Recognition Architectures”. *Neural Networks*, Vol. 3, pp.129-152, 1990.

[Carpenter92] Carpenter G., Grossberg S., Markuzon N., Reynolds J.H., Rosen D.B. “Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps”. *IEEE Transactions on Neural Networks*, Vol. 3.No.5, pp.698-713, 1992.

5 DYNAMIC TIME WARPING

Dynamic Time Warping (DTW) es una *función recursiva* (**6**), que *minimiza* la *distancia base* (**7**) entre dos secuencias (**5**); comúnmente la distancia base es la distancia euclidiana.

Otras definiciones de DTW pueden ser encontradas en [Keogh02] [Yi98] [Deller00]. Es así que la métrica base, es la función de semejanza entre un par de secuencias, series de tiempo, arreglos, etc.

A continuación se muestra el algoritmo de DTW.

$$Q = \left\langle q_1, q_2, \ldots, q_{|Q|} \right\rangle,\ C = \left\langle c_1, c_2, \ldots, c_{|C|} \right\rangle \tag{5}$$

$$DTW(\langle\ \rangle, \langle\ \rangle) = 0$$
$$DTW(Q, \langle\ \rangle) = DTW(\langle\ \rangle, C) = \infty$$
$$DTW(Q,C) = D_{base}(h(Q), h(C)) + \min \begin{cases} DTW(Q, r(C)) \\ DTW(r(Q), C) \\ DTW(r(Q), r(C)) \end{cases} \tag{6}$$

$$D_{base}(q_i, c_i) = \sqrt{(q_i - c_i)^2} \tag{7}$$

De esta forma se dice que:

$$DTW(Q,C) \to \infty, Q\ y\ C\ son\ diferentes$$
$$DTW(Q,C) = 0,\ \ Q\ y\ C\ son\ iguales$$

Las expresiones (**5**), (**6**), y (**7**) son extraídas de [Kim01], resulta ser la definición más clara y completa que existe de DTW*.

* $h(Q)$ Quitar el primer elemento de la secuencia Q; $|Q|$ Denota la longitud de la secuencia Q, $r(Q)$ Copia todos los elementos de la secuencia Q, a excepción del primero; $\langle\ \rangle$ Denota una secuencia vacía

A continuación se muestra el código fuente para DTW.

```
Objetivo: Calcular la distancia de semejanza entre las secuencias Q
y C, utilizando el algoritmo de Dynamic Time Warping
Parámetros: índices de las secuencias Q y C respectivamente
Salida: Distancia de semejanza (valor escalar) entre Q y C
```

```
double CDTW::DTW(int i, int j)
{
    if (i==Q.GetSize() && j==C.GetSize())
          return 0;
    else if ( (i==Q.GetSize()) | (j==C.GetSize()) )
          return 1.79769e+308;
    else
    {
          return sqrt(pow((Q.ElementAt(i)-C.ElementAt(j)),2)) +
          Minimum(DTW(i,j+1), DTW(i+1,j), DTW(i+1,j+1));
    }
}
```

La función recursiva de `DTW` es implementada como un método de la clase `CDTW`; éste requiere de dos parámetros, *i* y *j*, que son los índices de las secuencias Q y C, respectivamente (**4**). `Q` y `C` son objetos que pertenecen a la clase CArray de MFC [Pandolfi96], el método `GetSize()` devuelve el número de elementos contenidos en el arreglo, el método `ElementAt(i)` devuelve el valor almacenado en la *i*-ésima posición del arreglo.

Por último, se sustituyó ∞ por el valor máximo que puede ser asignado a una variable de tipo `double` (`1.79769e+308`). La figura 16, ilustra lo anterior. La versión completa del código se encuentra en el anexo A.

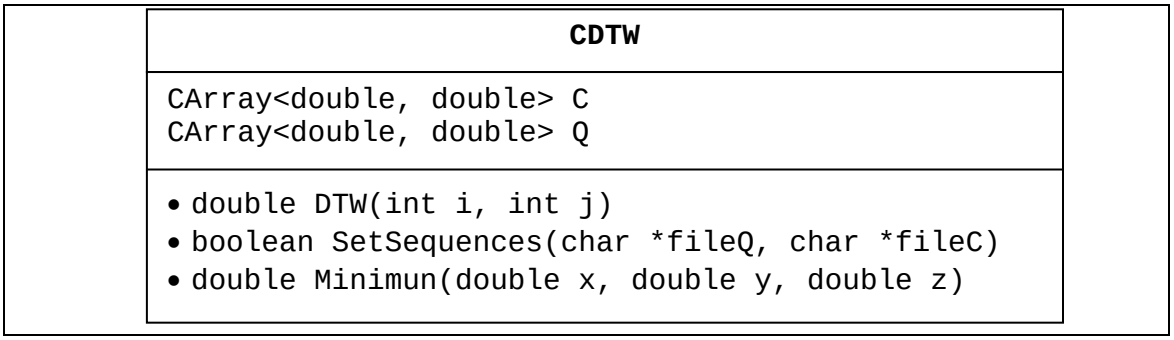


Figura 16. La clase CDTW compuesta por dos variables miembro Q y C, dos métodos miembro auxiliares y un método miembro principal (DTW).

Ahora, supongamos que $Q = \langle 1,1,1,1,1.5,2,2,1.5,1 \rangle$ y $C = \langle 1,1,1,1.5,1.5,2,3,1.5,1 \rangle$, son dos secuencias que se desean *comparar*, es decir, se requiere determinar una relación de semejanza.

En la figura 17, se muestra una matriz de distancias euclidianas, en la diagonal se encuentra la distancia euclidiana entre *Q* y *C*. Y el "camino" mínimo compuesto por ✧, es la distancia total de semejanza entre las secuencias.

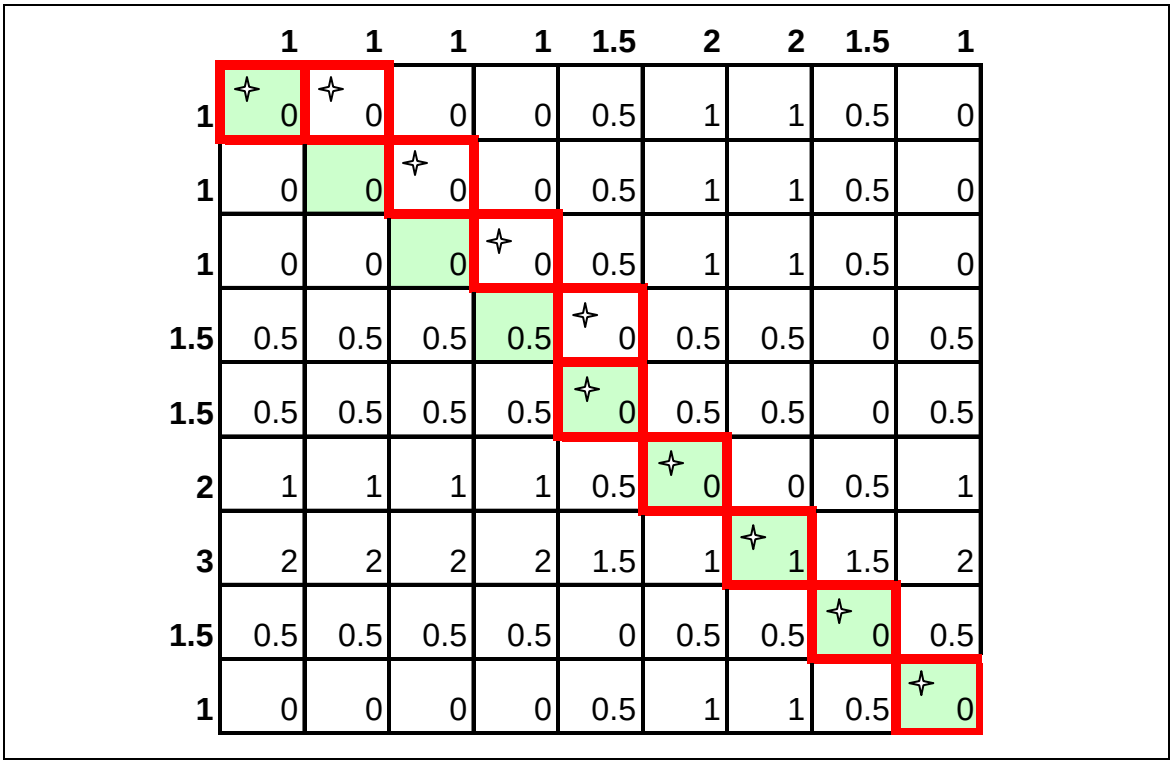


Figura 17. Comparación de dos secuencias. La distancia de DTW siempre será menor o igual a la distancia base localizada en la diagonal de la matriz. Nótese que la matriz sólo es explicativa del código de DTW, ya que dicha matriz no tiene que ser creada forzosamente.

DTW permite comparar secuencias de longitud variable, puede ser empleada para algunos problemas que involucran la comparación de secuencias pequeñas. Varias etiquetas han sido usadas para *describir* lo que hace DTW.

Alinear, estrechar, expandir, etc., han sido metáforas usadas para definir DTW, sin embargo, DTW no alinea, ni encoge, ni mucho menos expande secuencias para permitir su comparación. DTW determina una distancia mínima entre dos secuencias.

5.1 COMPLEJIDAD COMPUTACIONAL DE DTW

En [Keogh02, pág. 408] se hace la siguiente aseveración sobre la complejidad de DTW: "*The time and space complexity of DTW is O(nm).*" Donde *n* y *m* son las longitudes de dos secuencias.

Para demostrar que la complejidad de DTW no es de O(*nm*), realizamos una modificación al código de DTW.

La modificación consiste en agregar un contador (`count`), éste incrementa su valor cada vez que el código de DTW es invocado (llamado).

A continuación se muestra el código de DTW modificado.

```
Objetivo: Calcular la distancia de semejanza entre las secuencias Q
y C, utilizando el algoritmo de Dynamic Time Warping. Además contar
el número de veces que la función recursiva DTW es llamada
Parámetros: índices de las secuencias Q y C respectivamente
Salida: Distancia de semejanza (valor escalar) entre Q y C
```

```
double CDTW::DTW(int i, int j)
{
    count++;
    if (i==Q.GetSize() && j==C.GetSize())
          return 0;
    else if ( (i==Q.GetSize()) | (j==C.GetSize()) )
          return 1.79769e+308;
    else
    {
          return sqrt(pow((Q.ElementAt(i)-C.ElementAt(j)),2)) +
          Minimum(DTW(i,j+1), DTW(i+1,j), DTW(i+1,j+1));
    }
}
```

En la figura 18 se muestra una gráfica en la que se comparan pares de secuencias de la misma longitud, según [Keogh02] la complejidad temporal de DTW es de $O(n^2)$, donde n es la longitud de las secuencias en cuestión.

En la figura 18, esta complejidad temporal "teórica" se ilustra con una línea marcada por cuadros; la complejidad temporal real de DTW se ilustra con la línea marcada por círculos, en el eje x se encuentra la longitud de las secuencias comparadas, en el eje y se denota el número de llamadas o invocaciones al código de DTW dado un cierto tamaño de secuencias.

Nótese la escala del eje de *Número de Llamadas*.

La pregunta que inmediatamente se debe realizar es, ¿por qué crece exponencialmente la complejidad temporal del algoritmo?, esta interrogante será explorada más adelante dentro del capítulo.

La siguiente pregunta es, ¿qué se puede hacer para reducir la complejidad temporal de *DTW*?, es aquí donde se han propuesto técnicas para reducir la complejidad de *DTW*.

A continuación se muestran cuatro principales enfoques, *Métricas* $\leq$ *DTW*, *Áreas Restringidas*, *Reducción de Dimensionalidad*, e *Indexado*.

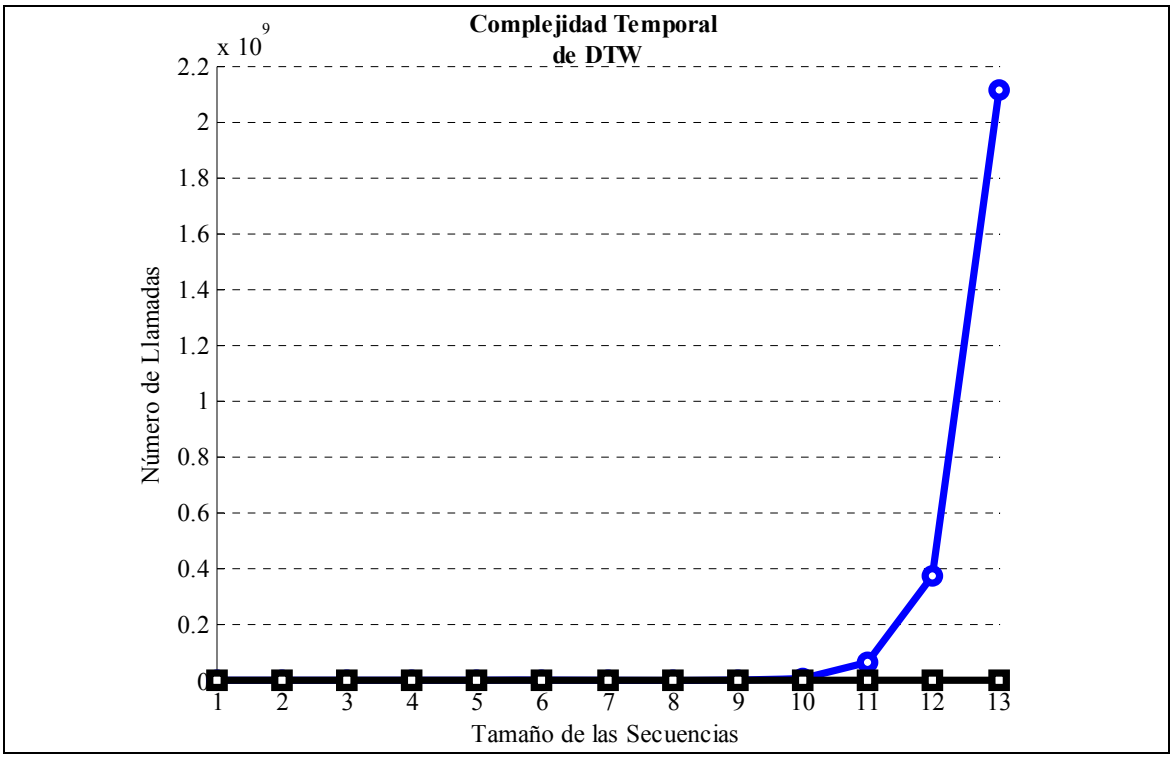


Figura 18. Complejidad Temporal. La línea marcada con cuadros denota crecimiento de $O(n^2)$, donde n es el tamaño del par de secuencias comparadas. La línea marcada con círculos denota el crecimiento real (exponencial) del código de DTW. Nótese la escala del eje y.

5.1.1 MÉTRICAS ≤ DTW

Existen tres métricas ampliamente usadas en el problema de semejanza, LB_Yi [Yi98], LB_Kim [Kim01], y LB_Keogh [Keogh01], LB denota *Lower Bounding*.

Las estrategias que siguen estas métricas son:

1. Considerar sólo ciertos elementos de un par de secuencias dadas, de tal forma que la métrica mida alguna propiedad deseada. El proceso o algoritmo para medir la(s) propiedad(es) deseada(s), se precisa que sea de complejidad computacional menor a la de DTW.

2. Utilizar la nueva métrica como cedazo, es decir, eliminar secuencias que no son consideradas candidatas a ser similares de acuerdo a DTW.

La métrica LB_Yi utiliza sólo los valores máximos y mínimos de un par de secuencias dado. Las duplas son utilizadas para calcular una distancia de semejanza entre secuencias.

La idea que lo sustenta es que los valores máximos y mínimos de una secuencia son también considerados por DTW y por lo tanto LB_Yi ≤ DTW.

Se ha demostrado que la métrica LB_Yi, no garantiza descartar falsos candidatos. Demostración de lo anterior se encuentra en [Berndt96].

LB_Kim a diferencia de LB_Yi, logra garantizar descartar falsos candidatos. Con ello permite la utilización de estructuras de datos multidimensionales como los R-Tree y sus variantes [Guttman84].

LB_Kim considera cuatro elementos por cada secuencia, el máximo, mínimo, primer, y último elemento de una secuencia.

De forma análoga a LB_Yi, demuestra en [Kim01] que LB_Yi ≤ DTW.

A continuación se muestra la definición de LB_Kim.

$$LB\_Kim(Q,C) = \max\begin{cases} |First(Q) - First(C)| \\ |Last(Q) - Last(C)| \\ |Greatest(Q) - Greatest(C)| \\ |Smallest(Q) - Smallest(C)| \end{cases} \quad (\mathbf{8})$$

Por último, Keogh en [Keogh01] introduce una nueva métrica LB_Keogh, esta métrica no considera solamente algunos elementos de las secuencias, como lo hace LB_Yi y LB_Kim, LB_Keogh argumenta la utilidad de su métrica basándose en envolventes de cada secuencia a comparar.

Estas envolventes son tomadas de un enfoque que se describirá más adelante (*Áreas Restringidas de una Matriz*).

A continuación se muestra la definición de LB_Keogh.

$$LB\_Keogh(Q,C) = \sqrt{\sum_{i=1}^{n} \begin{cases} (c_i - U_i)^2 & si\ c_i > U_i \\ (c_i - L_i)^2 & si\ c_i > L_i \\ 0 & De\,otra\ forma \end{cases}} \quad (\mathbf{9})$$

$$\begin{aligned} U_i &= \max(q_{i-r} : q_{i+r}) \\ L_i &= \min(q_{i-r} : q_{i+r}) \end{aligned} \quad (\mathbf{10})$$

5.1.2. ÁREAS RESTRINGIDAS

Para reducir la complejidad computacional de DTW algunos autores sugieren, la creación de una matriz de $n \times m$, donde n y m son las longitudes de las secuencias a comparar.

Cada celda i,j representa la distancia base (**7**) entre el i-ésimo elemento de la primer secuencia y el j-ésimo elemento de la segunda.

Este enfoque argumenta que sólo un área restringida de la matriz interviene en el costo mínimo de un "camino", que es determinado por DTW. Estas áreas restringidas son las restricciones globales de DTW.

Existen dos restricciones globales de DTW, el paralelograma de Itakura, y la banda de Shakoe-Chiba [Deller00]. En la figura 19 se muestran estas restricciones.

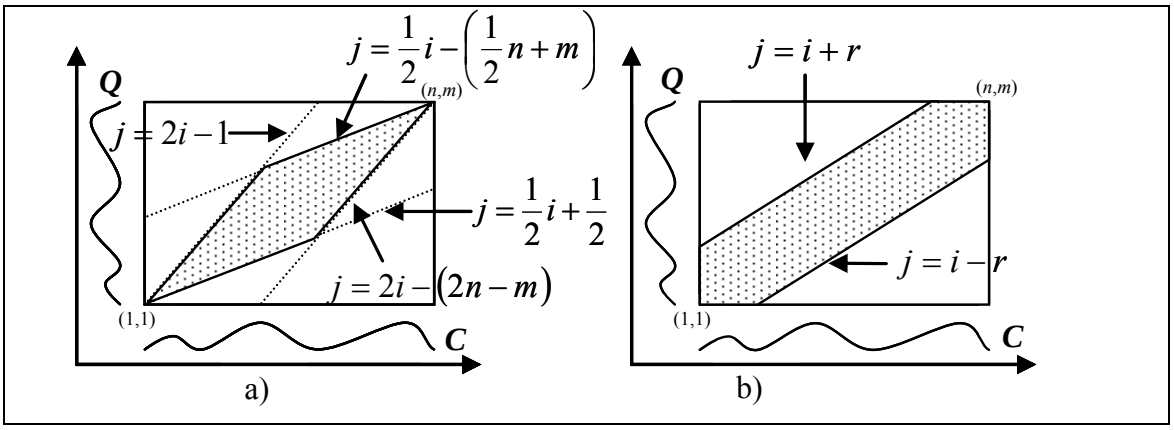


Figura 19. a) Paralelograma de Itakura, y b) Banda de Shakoe-Chiba, restricciones globales de DTW. Esquemas extraídos de [Deller00].

Se debe mencionar que estas restricciones globales definen áreas simétricas, algo que es útil cuando las secuencias son semejantes, es decir, la distancia mínima se compone por celdas i,j que están cerca de la diagonal de la matriz.

En [Angeles05e] se propone modificar las fórmulas para el paralelograma de Itakura, ya que los índices deben ser enteros y no reales. A continuación se muestra la corrección de estos índices en (**11**).

Aunque el problema radica determinar áreas restringidas irregulares, esto es, que dependa del par de secuencias a comparar, y que el cálculo de esta área sea de una complejidad menor o igual a la complejidad de DTW.

$$j = \begin{cases} \left\lfloor \frac{1}{2} i + \frac{1}{2} \right\rfloor, & i \leq \left\lceil \frac{n}{2} \right\rceil \\ 2i - (2n - m), & i > \left\lceil \frac{n}{2} \right\rceil \end{cases} \quad (11)$$

Aún se discute la repercusión de las restricciones globales en el desempeño y precisión en el cálculo de DTW.

5.1.3. REDUCCIÓN DE DIMENSIONALIDAD

Otra opción que se ha propuesto para reducir el tiempo de ejecución de DTW es la de reducir la dimensionalidad de las secuencias a comparar, se reduce el tiempo de ejecución ya que son menos los elementos de la secuencia original los que se ven involucrados, sin embargo, la complejidad de DTW no se modifica.

Algunas de las técnicas para la reducción de dimensionalidad se pueden encontrar en [Agrawal93], [Chan02], [Korn97], [Keogh02], [Morinaka01], y [Lin03].

Cada una de las técnicas preserva alguna propiedad de una secuencia dada, la eficacia de dicha técnica depende de la "naturaleza" de los datos, no todas sirven para un caso dado.

Aunque ha tenido un gran auge la aplicación de técnicas de reducción de dimensionalidad para el manejo de grandes bases de datos, ellas no atacan directamente el problema de la complejidad de DTW.

5.1.4. INDEXADO

Cuando las métricas y técnicas de reducción de dimensionalidad postularon la posibilidad de representar una secuencia de p elementos en un espacio k-dimensional, y que ésta representación preserve alguna propiedad deseada, surgió la posibilidad de utilizar las estructuras multidimensionales como R-Tree [Guttman84] y sus variantes.

El objetivo de indexar las secuencias es el proponer un orden dentro de la estructura multidimensional, y así, realizar búsquedas no-secuenciales.

Sin embargo, la complejidad computacional de DTW no cambia, sólo se reduce el número de candidatos para una consulta dada.

	Reducir Complejidad de DTW
Métricas ≤ DTW	*Objetivo:* Crear métricas que sean menor o igual a la distancia de DTW, pero de menor complejidad computacional. *Problema:* El orden parcial/total que se determina con DTW y con la nueva métrica no necesariamente el mismo para un conjunto de secuencias dado. Además, se requiere de calcular DTW con los candidatos posibles.
Áreas Restringidas de una Matriz	*Objetivo:* Definir áreas válidas donde un camino es posible candidato de ser el camino mínimo. *Problema:* Determinar áreas válidas no simétricas en tiempo polinomial.
Reducción de Dimensionalidad	*Objetivo:* Representar un ente de dimensional p en un espacio k-dimensional, donde $k<p$, y ésta representación conserve algunas propiedades deseadas. *Problema:* El reducir la longitud de las secuencias no modifica la complejidad de DTW.
Indexado	*Objetivo:* Ser una aproximación a la búsqueda secuencial, sin examinar todos los elementos de un conjunto. *Problema:* Al igual que la *Reducción de Dimensionalidad*, la complejidad de DTW no cambia.

Aunque estos cuatro enfoques han impulsado en gran medida a atacar el problema de semejanza, no atacan directamente la complejidad computacional de DTW, ya que todos los enfoques consideran en algún momento la necesidad de calcular DTW.

Ante lo anterior, retomamos la pregunta:

¿Por qué crece exponencialmente la complejidad temporal del algoritmo (DTW)?

5.2 FAST DYNAMIC TIME WARPING

Para contestar dicha pregunta se presenta un algoritmo usado en [Jang00].

```
Objetivo: Calcular la distancia de semejanza entre las secuencias Q
y C, utilizando el algoritmo de Fast Dynamic Time Warping
Parámetros: Secuencias Q y C
Salida: Distancia de semejanza (valor escalar) entre Q y C
```

```
double CFDTW::FDTW() {
    // Construir la tabla de DTW
    Matrix = new double*[Q.GetSize()];
    for(int i=0; i < Q.GetSize(); i++)
          Matrix[i] = new double [C.GetSize()];

    // Primer elemento de la Matrix
    Matrix[0][0] = sqrt(pow(Q.ElementAt(0)-C.ElementAt(0), 2));

    // Construir la primer fila de la tabla DTW
    for(int j=1; j<C.GetSize(); j++)
          Matrix[0][j] = Matrix[0][j-1] + sqrt(pow(Q.ElementAt(0)-
                                C.ElementAt(j), 2));

    // Construir la primer columna de la tabla DTW
    for(i=1; i<Q.GetSize(); i++)
          Matrix[i][0] = Matrix[i-1][0] + sqrt(pow(Q.ElementAt(i)-
                                          C.ElementAt(0), 2));

    // Construir el resto de FILAS de la tabla DTW
    for(i=1; i<Q.GetSize(); i++)
          for(j=1; j<C.GetSize(); j++) {
                // Obtener la distancia entre puntos (Q y C)
                double pdistance = sqrt(pow(Q.ElementAt(i)-
                                          C.ElementAt(j), 2));
                Matrix[i][j] = Matrix[i-1][j-1] + pdistance;

                if ( (Matrix[i-1][j]+pdistance) < Matrix[i][j] )
                      Matrix[i][j] = Matrix[i-1][j] + pdistance;

                if ( Matrix[i][j-1]+pdistance < Matrix[i][j])
                      Matrix[i][j] = Matrix[i][j-1] + pdistance; }
    return Matrix[Q.GetSize()-1][C.GetSize()-1];  }
```

El método `FDTW` es parte de la clase `DTW`, no requiere de parámetros, sólo de las secuencias a comparar (`Q` y `C`), que son objetos que pertenecen a la clase `DTW`, el algoritmo construye una matriz de $|Q|\times|C|$, la primera celda de la matriz (0,0), contendrá la distancia base entre los primeros elementos de las secuencias. El siguiente paso es calcular las distancias para la primera fila de la matriz, a partir de (0,1), ya que las distancias serán entre los elementos q_0 (elemento "*fijo*") y c_j de las secuencias. De forma

similar el siguiente ciclo es para determinar la primera columna de la matriz, que consiste de las distancias entre los elementos c_0 (ahora, elemento "fijo") y q_i de las secuencias.

Finalmente, para construir el resto de las filas de la tabla, se requiere de determinar a priori la distancia entre los elementos q_i y c_j, ésta distancia (distancia correspondiente a la celda (i,j)) será aumentada con la distancia del vecino (($i-1,j-1$), ($i-1,j$), ($i,j-1$)) cuya suma la sea superior.

La complejidad espacial de FDTW es de |Q|×|C|, la complejidad temporal de FDTW es de $O(|C-1|) + O(|Q-1|) + O(|C-1|*|Q-1|)$.

Una de las grandes diferencias que tiene FDTW con respecto a DTW es su capacidad de determinar un "camino" mínimo de forma no-recursiva.

Supóngase, de nuevo que se tienen dos secuencias, $Q = \langle 1,1,1,1,1.5,2,2,1.5,1 \rangle$ y $C = \langle 1,1,1,1.5,1.5,2,3,1.5,1 \rangle$, las mismas secuencias involucradas en la figura 17.

A continuación se muestra una matriz, en la cual se calculó una distancia base mínima entre Q y C.

	1	**1**	**1**	**1**	**1.5**	**2**	**2**	**1.5**	**1**
1	0	0	0	0.5	1	2	4	4.5	4.5
1	**0**	**0**	**0**	**0.5**	1	2	4	4.5	4.5
1	0	0	0	0.5	1	2	4	4.5	4.5
1.5	0	0	0	0.5	1	2	4	4.5	4.5
1.5	0.5	0.5	0.5	0	0	0.5	2	2	2.5
2	1.5	1.5	1.5	0.5	0.5	0	1	1.5	2.5
3	2.5	2.5	2.5	1	1	0	1	1.5	2.5
1.5	3	3	3	1	1	0.5	1.5	1	1.5
1	3	3	3	1.5	1.5	1.5	2.5	1.5	1

Figura 20. Matriz calculada con FDTW, nótese que la distancia mínima entre dos secuencias dadas, siempre se localizará en la celda ($|Q|,|C|$), señalada por la flecha en la figura.

REFERENCIAS DEL CAPÍTULO

[Koegh02] Keogh E. "Exact Indexing of Dynamic Time Warping". *In Proceedings of the 28th VLDB Conference*, pp. 406-417, 2002.

[Yi98] Yi B. K., Jagadish H. V., Faloutsos C. "Efficient Retrieval of Similar Time Sequences Under Time Warping". *In Proceedings of the 14th International Conference on Data Engineering (ICDE'98)*, pp. 201-208, IEEE Computer Society Press, 1998.

[Deller00] Deller J. R., Hansen J. H. L, Proakis J. G. *Discrete-Time Processing of Speech Signals*. Wiley-IEE Press, ISBN: 0780353862, Second Edition, pp 623-676, 2000.

[Kim01] Kim S. W., Park S., Chu W. W. "An Index-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Database". *In Proceedings of the 17th International Conference on Data Engineering (ICDE'01)*, pp. 607-614, 2001.

[Pandolfi96] Pandolfi F., Oliver M., Wolski M. *Microsoft Foundation Class 4 Bible*. Waite Group Press, ISBN: 1571690212, 1996.

[Berndt96] Berndt D. J., Clifford J. "Finding Patterns in Time Series: A Dynamic Programming Approach". *Advances in Knowledge Discovery and Data Mining*, AAAI/MIT, pp. 229-248, 1996.

[Guttman84] Guttman A. "R-Trees: A Dynamic Index Structure for Spatial Searching". *In Proceedings of ACM SIGMOD'84 Conference on Management of Data*, ACM Press, pp. 47-57, 1984.

[Angeles05e] Angeles-Yreta A., Figueroa-Nazuno J., Ramírez-Amaro K. "Búsqueda de Semejanza entre Objetos 3D por Indexado". *Decimosexta Reunión de Otoño Comunicaciones, Computación, Electrónica y Exposición Industrial*, ROC&C' 2005.

[Agrawal93] Agrawal R., Faloutsos C., Swami A. "Efficiency Similarity Search in Sequence Databases". *In Proceedings of the 4th Conference of Foundations of Data Organization and Algorithms (FODO'93)*, Lecture Notes in Computer Science, pp. 69-84, 1993.

[Chan02] Chan K.P., Fu A., Yu C. "Haar wavelets for efficient similarity search of time-series with and without time warping". *IEEE Transactions on Knowledge and Data Engineering, to appear*, 2002.

[Korn97] Korn F., Jagadish H., Faloutsos C. "Efficiently supporting ad hoc queries in large datasets of time sequences". *In Proceedings of ACM SIGMOD'97*, pp. 289-300, 1997.

[Keogh01] Keogh E., Chakrabarti K., Pazzani M., Mehrotra. "Locally adaptive dimensionality reduction for indexing large time series databases". *In Proceedings of ACM SIGMOD'01 Conference on Management of Data*, pp. 151-162, 2001.

[Morinaka01] Morinaka Y., Yoshikawa M., Amagasa T., Uemura S. "The L-index: An Indexing Structure for Efficient Subsequence Matching in Time Sequence Databases". *In Proceedings of Pacific-Asian Conference on Knowledge Discovery and Data Mining*, PAKDD'01, pp. 51-60, 2001.

[Lin03] Lin J., Keogh E., Lonardi S., Chiu B. "A Symbolic Representation of Time Series, with Implications for Streaming Algorithms". *In Proceedings of the 8th ACM SIGMOD'03*, Workshop on Research Issues in Data Mining and Knowledge Discovery. pp. 2-11, 2003.

[Jang00] Jang J.S.R, Gao M.Y. "A Query-by-Singing System based on Dynamic Programming". *International Workshop on Systems Resolutions* (the 8th Bellman Continuum), pp. 85-89, 2000.

6 EL CONJUNTO DE DATOS

La base de datos de figuras geométricas usada en este trabajo se compone de dos conjuntos de objetos:

a) Las figuras geométricas creadas a partir de plantillas predefinidas por un sistema de DAC típico (sección *Creación de Figuras con 3D Studio MAX*).

b) Las figuras geométricas creadas a partir de figuras *base* (sección *Creación de Figuras seudo-aleatorias*).

Posteriormente, se presentan resultados ya publicados, de las técnicas presentadas en capítulo de resultados.

6.1 CREACIÓN DE FIGURAS GEOMÉTRICAS CON 3D STUDIO MAX

3D Studio MAX es un sistema de DAC comercial, generalmente es usado para la creación de gráficos 3D. Al igual que otros paquetes similares, 3D Studio MAX, proporciona plantillas predefinidas para crear objetos básicos, como cajas, conos, esferas, cilindros, pirámides, etc.

Estos objetos son modificables vía parámetros. El número de parámetros y el rango de éstos, está en función del objeto en cuestión.

En la figura 21, se muestra la creación de una caja o cubo usando las platillas predefinidas de 3D Studio MAX, los parámetros para este objeto son longitud, anchura, altura, y el respectivo número de segmentos.

Cada vez que se crea una nueva figura geométrica, se almacena en formato VRML97 [Hartman96], la gramática de dicho formato contempla el arreglo de vértices e índice de vértices, que se utilizan para representar y determinar semejanza en este trabajo.

Cabe mencionar que dichas estructuras se encuentran prácticamente en todos los formatos para objetos 3D.

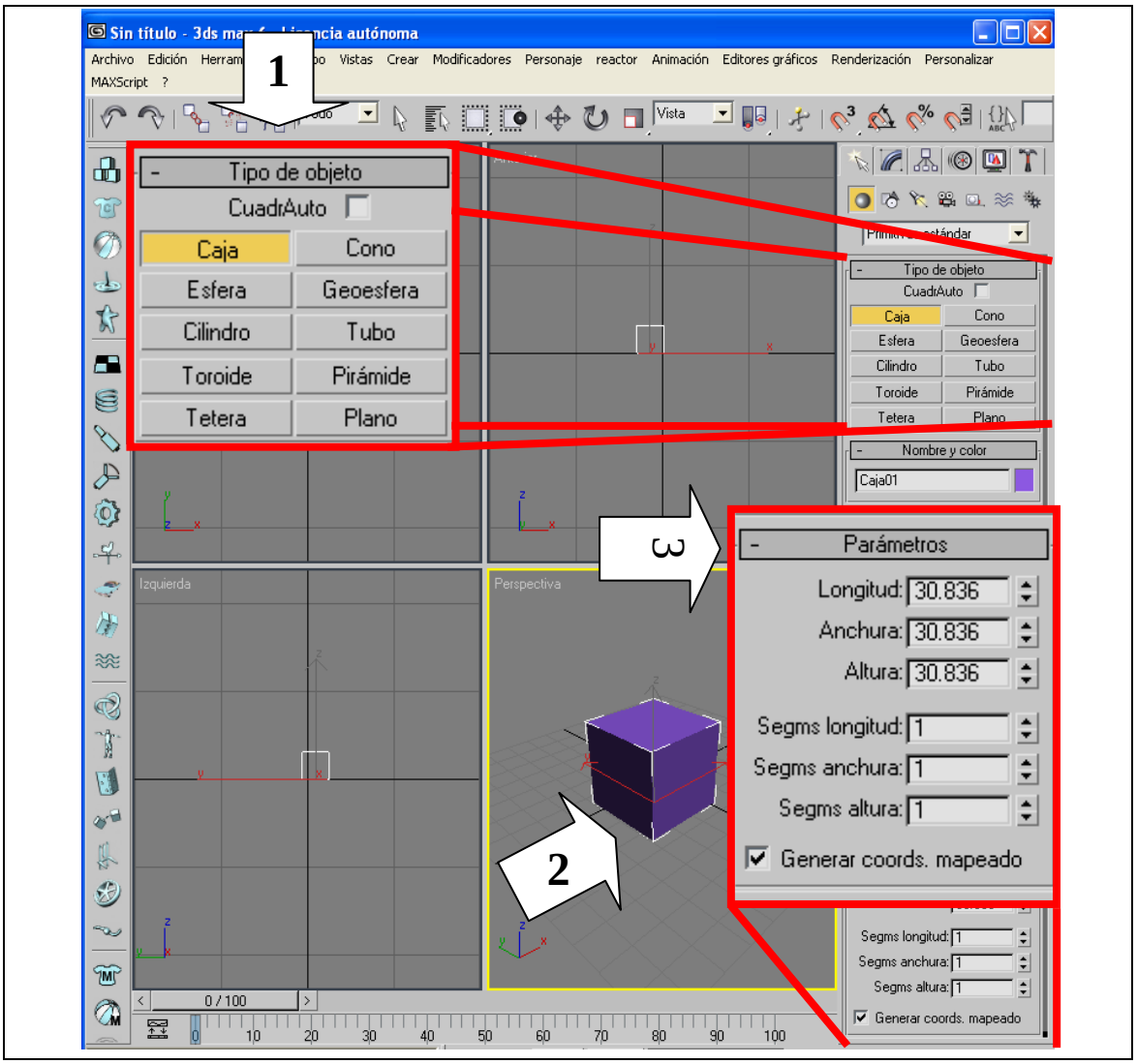


Figura 21. Creación de una figura geométrica (cubo) usando las plantillas predefinidas de 3D Studio MAX. El procedimiento es: seleccionar el tipo de objeto que se desea crear (flecha 1), colocarlo en uno de los cuadrantes (flecha 2), y modificar los parámetros disponibles (flecha 3).

En la figura 22, se ilustra el procedimiento para exportar objetos 3D de 3D Studio MAX a archivos con formato VRML97 [Hartman96].

En este trabajo se creó una base de datos de figuras geométricas por medio de las platillas predefinidas de 3D Studio MAX. Por cada tipo de objeto (plantilla), se crean diferentes figuras geométricas, modificando los parámetros disponibles.

Para el primer experimento se crearon 46 figuras geométricas, extendiéndose hasta 62 figuras geométricas en experimentos posteriores. Esta base de datos es representativa de objetos 3D creados por sistemas de DAC.

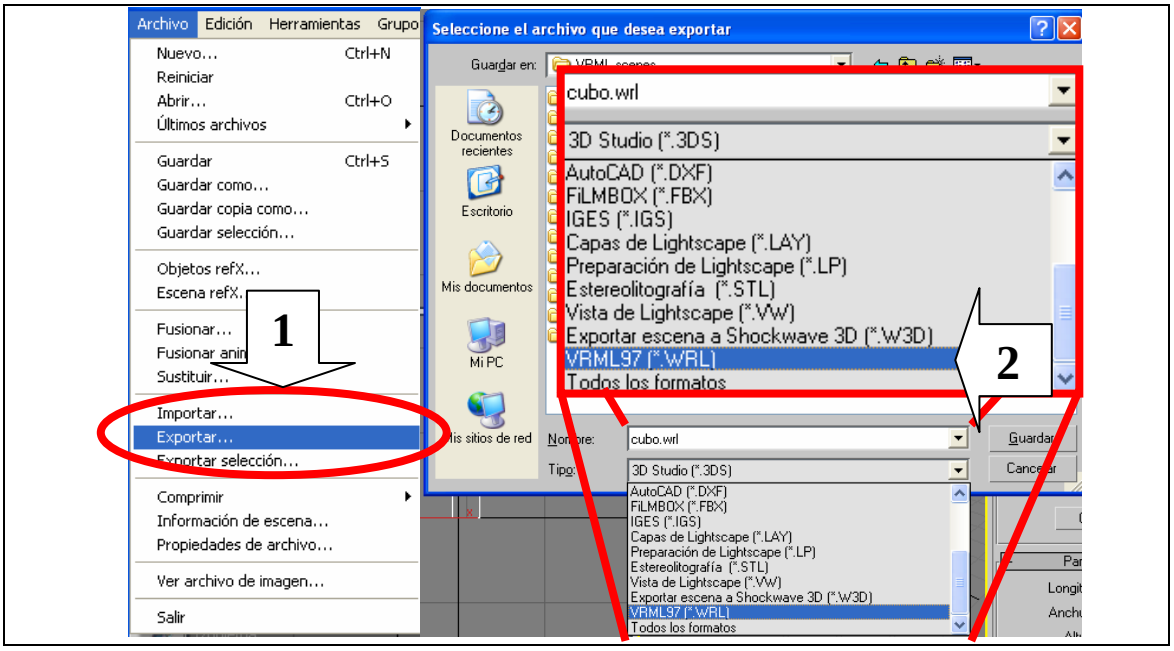


Figura 22. Exportar objetos 3D de 3D Studio MAX a archivos con formato VRML97. Una vez creada la figura en 3D Studio MAX, seleccionar la opción de Exportar en el menú de Archivo (flecha 1), proporcionar un nombre de archivo y guardar con formato VRML97 (flecha 2).

6.2 CREACIÓN DE FIGURAS GEOMÉTRICAS SEUDO-ALEATORIAS

Las figuras seudo-aleatorias son creadas a partir de figuras *base* (cubos, pirámides, etc.), el objetivo es crear una base de datos de figuras geométricas, en la cual se desconozca clasificación alguna de antemano [Makoto98]. A continuación se muestra el algoritmo para dicha tarea.

```
Procedimiento: Crear figuras geométricas seudo-aleatorias
Parámetros: K figuras base, N (número de figuras seudo-
aleatorias por cada figura base que se desea crear)
Salida: N*K figuras geométricas seudo-aleatorias
```

```
Por cada figura base_k, hacer
  Obtener el arreglo de vértices de la figura base_k
  De 1 a N, hacer
    Determinar un índice de modificación (mx)
    Seleccionar algún tipo de modificación (A, B ó C)
    De 1 a mx, hacer
      Calcular un valor seudo-aleatorio i en el rango
      [1..longitud del arreglo de vértices]
       Modificar el vértice i, con el tipo de modificación
       seleccionada
    Fin
    Almacenar la enésima figura geométrica seudo-aleatoria
  Fin
Fin
```

El algoritmo anterior creará `N*K` figuras geométricas seudo-aleatorias, donde `N` es el número de figuras seudo-aleatorias por cada figura base que se desea generar, y `K` es número de figuras base.

El arreglo de vértices es la única estructura de la figura base que es afectada. El índice de modificación *mx*, es el número de vértices que deben ser modificados, este valor puede ser fijo o determinado de forma seudo-aleatoria. Los tipos de modificación son heurísticas, el propósito es crear figuras geométricas *complicadas*, pero que puedan conservar alguna estructura, capaz de ser detectada por el criterio humano.

Por último, almacenar la nueva figura seudo-aleatoria (también en formato VRML97), involucra repetir el archivo de la figura base a excepción del arreglo de vértices modificado. En la figura 23, se muestran tres figuras seudo-aleatorias creadas a partir de una figura base, en este caso, un cubo.

A continuación se detallan 3 heurísticas para la modificación de vértices (`modificación tipo A, B, y C`).

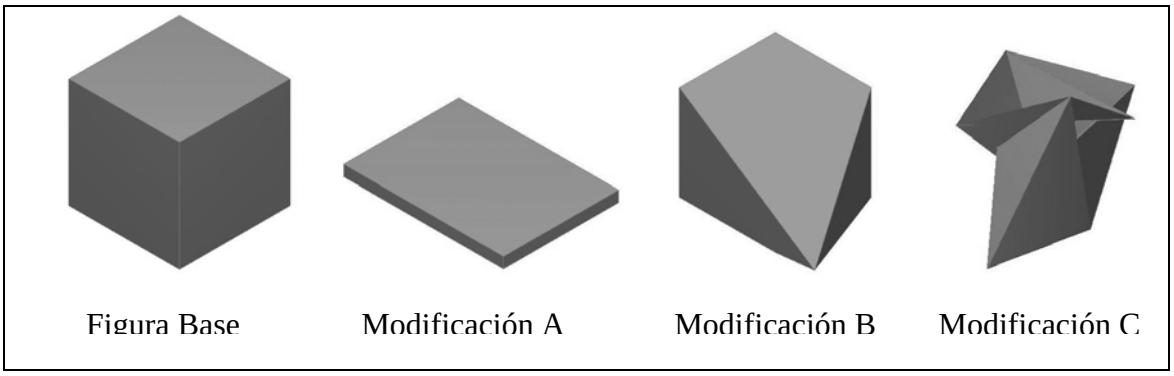


Figura 23. *Figura Base y tres tipos de modificaciones*. El arreglo de vértices de la figura base es modificado, dependiendo del tipo de modificación, la figura seudo-aleatoria resultante conserva en mayor o menor medida la estructura de la figura base.

Procedimiento: Modificación tipo A **Parámetros:** *i* (índice del vértice a modificar) **Salida:** Vértice modificado
Determinar nuevos valores para *x*, *y*, y *z*, (_x, _y, _z) Preservar signos del vértice *i* Actualizar todas las incidencias del vértice *i* en el arreglo de vértices

Procedimiento: Modificación tipo B **Parámetros:** *i* (índice del vértice a modificar) **Salida:** Vértice modificado
Determinar nuevos valores para *x*, *y*, y *z*, (_x, _y, _z) Preservar signos del vértice *i* Sustituir valores *x*, *y*, y *z* del vértice *i* con _x, _y, y, _z

Procedimiento: Modificación tipo C **Parámetros:** *i* (índice del vértice a modificar) **Salida:** Vértice modificado
Determinar nuevos valores para *x*, *y*, y *z*, (_x, _y, _z) Sustituir valores *x*, *y*, y *z* del vértice *i* con _x, _y, y, _z

Los tres tipos de modificaciones determinan de forma seudo-aleatoria nuevos valores *x*, *y*, y *z* para cada vértice a modificar.

La modificación A, preserva los signos de los escalares *x*, *y*, y *z*) a sustituir, y actualiza todas las incidencias del vértice modificado en el arreglo de vértices.

La modificación B, también preserva los signos de los escalares *x*, *y*, y *z* a sustituir, sin embargo, este tipo de modificación solo actualiza dichos valores para el vértice designado.

Por último, la modificación C, sólo actualiza valores de *x*, *y*, y, *z* del vértice seleccionado.

REFERENCIAS DEL CAPÍTULO

[Hartman96] Hartman, J., Wernecke, J. *The VRML 2.0 handbook: building moving worlds on the web*, *Addison-Wesley*, 1996.

[Makoto98] Makoto M., Takuji N. "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator". *ACM Transactions on Modeling and Computer Simulation*, ACM Press, Vol. 8, 1998, pp. 3-30.

7 REPRESENTACIÓN Y CÓMPUTO DE SIMILITUD USANDO RNA'S FUZZY ART

En este capítulo se abordan dos temas principales. 1) La representación de un figura geométrica como una tripleta de secuencias acoplada; dicha representación no utiliza los descriptores reportados en la literatura; el aspecto más importante de esta representación es la capacidad de abstraer un elemento tridimensional en un conjunto de secuencias, la cuales pueden ser convenientemente utilizadas para determinar una relación de semejanza con otra entidad tridimensional. Y 2) El cómputo de semejanza entre figuras geométricas usando la tripleta de secuencias acopladas y una tripleta de RNA's Fuzzy ART, éstas determinan una categoría por coordenada a cada figura del conjunto, finalmente se utiliza la distancia de Hamming para la unificación de dichas categorías.

7.1 REPRESENTACIÓN DE FIGURAS EN SECUENCIAS ACOPLADAS

Cualquier *entidad* tridimensional creada, manipulada, y/o almacenada en una computadora, es esencialmente un grafo *no-dirigido* en un *espacio* tridimensional. [Angeles04a] [Angeles05b] [Angeles05c] [Angeles05d] [Angeles05e], ver figura 24. Dichos grafos son codificados en una *especie* de matriz de adyacencia.

Esta *seudo matriz de adyacencia* consiste de dos *estructuras* principales:

1) ***Arreglo de vértices V***. Esta *estructura* contiene todos los vértices que componen al grafo no-dirigido, cada vértice se compone de una tripleta de valores escalares, que corresponden a las coordenadas en *X*, *Y*, y *Z*.

2) ***Índice de vértices F***. El índice de vértices es una *lista* que indica el *orden* en que deben ser *colocados* los vértices en el espacio tridimensional.

$$V = \langle v_1, v_2, \ldots, v_{|V|} \rangle, \text{ donde } v_i = (x_i, y_i, z_i) \text{ y } x_i, y_i, z_i \in \Re \quad \textbf{(12)}$$

$$F = \langle v_{f_1}, v_{f_2}, \ldots, v_{f_n} \rangle, \text{ donde } f_j \in [1, |V|] \quad \textbf{(13)}$$

Adicionalmente a estas dos estructuras, en ocasiones se definen matrices para la rotación, traslación y escalamiento de la entidad tridimensional; así como información acerca de la textura, perspectiva, iluminación, etc. [Segal04]. Para las figuras que en este trabajo se abordan se consideran objetos aislados y bien definidos, es decir, sólo se requiere de un arreglo de vértices (**12**) y un índice de vértices (**13**) para codificar la *geometría* de la figura geométrica en cuestión.

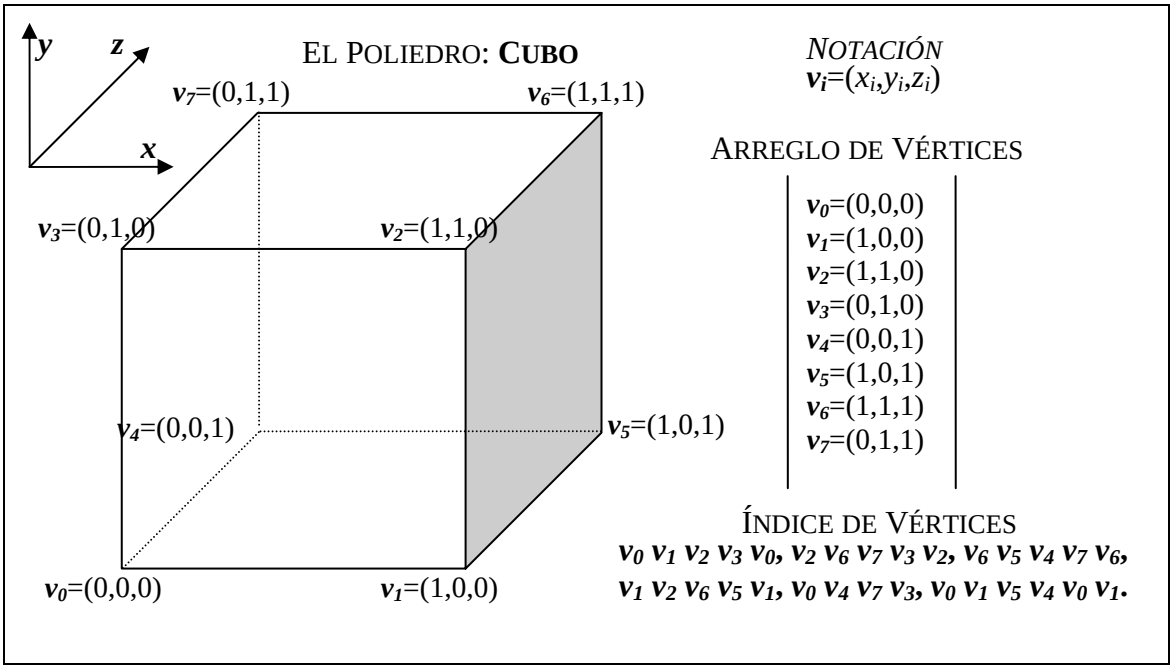


Figura 24. Representación de figuras geométricas, objetos 3D, modelos 3D como grafos no-dirigidos. Nótese que $x \in \Re, y \in \Re, z \in \Re$.

La figura de la figura 24, muestra un cubo y su abstracción como un grafo no-dirigido en un espacio tridimensional, el arreglo de vértices y el índice de vértices son estructuras indispensables para describir la geometría de los modelos 3D. Información adicional como texturas, sombras, cámaras, etc., es ignorada; en este trabajo, los métodos de semejanza entre figuras se enfocan sólo en la *forma geométrica* de las entidades.

Se consideran (**12**) y (**13**) como estructuras muy generales que permiten la definición de cualquier modelo 3D. La primera hipótesis que se formula en [Angeles04a] es que en esas dos estructuras (que pueden conformar una matriz), está toda la información que se requiere para la comparación de dos figuras, ya que ahí se encuentra definida la topología de las figuras geométricas.

De lo anterior se puede deducir que una ventaja de ello es que las figuras que se comparan utilizando dichas estructuras, no necesariamente tienen que ser sólidos. En la sección de resultados se muestran pruebas de esta aseveración.

Múltiples formatos de archivos contemplan en su gramática las estructuras (**12**) y (**13**), en la figura 25 se muestra un archivo ejemplo de una figura (piramide01.wrl) en formato *wrl*, que pertenece al estándar VRML 2.0 [R].

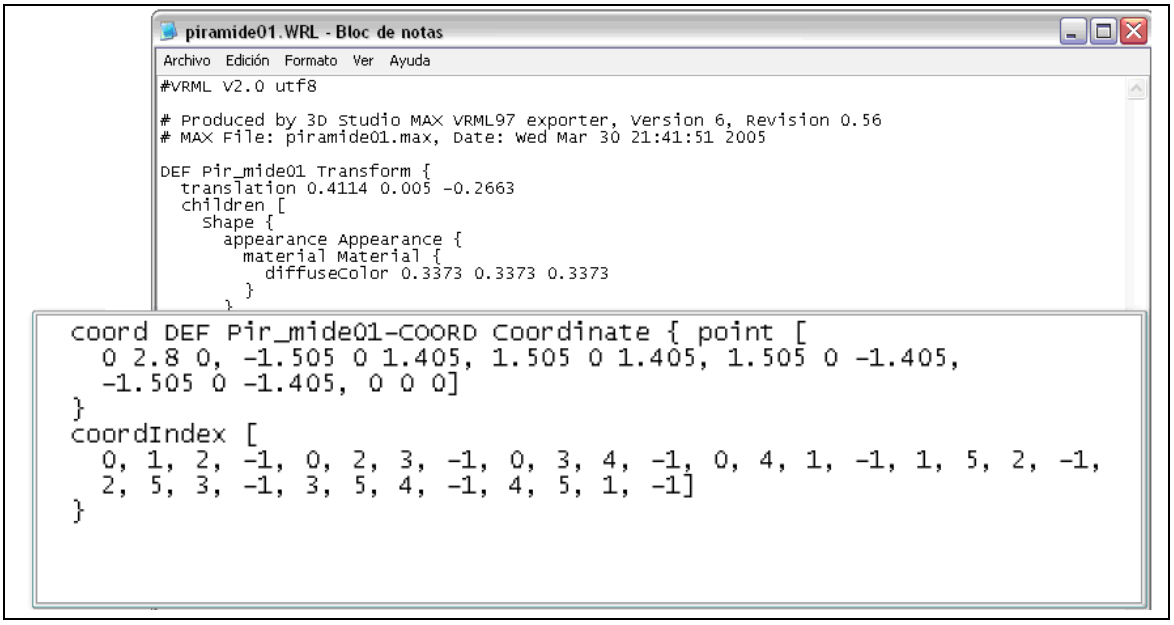


Figura 25. Archivo VRML 2.0 de una figura. Nótese que la palabra reservada `Coordinate` indica la definición del arreglo de vértices, la palabra reservada `coordIndex` indica la definición del índice de vértices, en éste la presencia de -1 indica el fin de una cara y la repetición del primer vértice dentro de la secuencia de la cara.

Como ya se ha mencionado, el arreglo de vértices define los nodos del grafo no-dirigido en el espacio tridimensional. Sin embargo, es el índice de vértices quien termina por definir la geometría de la figura en cuestión. Esta premisa es idea que fundamenta la representación de las figuras geométricas en secuencias acopladas.

A continuación se muestra un segmento de código en C++, en el que se utiliza primitivas de OpenGL (*Open Graphics Library*) para la graficación de *cualquier* objeto 3D [Segal04]. Detalles como inicialización del dispositivo gráfico, entre otras cosas han sido omitidos. El código completo se encuentra en el Anexo A. Existe una gran cantidad gramáticas que acuerdan la codificación de objetos 3D en arreglo de vértices e índice de estos.

```
Objetivo: Graficar un objeto 3D
Parámetros: Arreglo de vértices V e índice de vértices F
```

```
glBegin(GL_POLYGON);
  for (int i=0; i < length(F); i++)
    glVertex3f(V[F[i]].x, V[F[i].y, V[F[x].z);
glEnd();
```

En el programa anterior, el listado (`for`) de los vértices, en el *orden* que se ha establecido, especifica la geometría de la figura en cuestión. Este listado es la razón que sustenta la idea de representar a las figuras en secuencias acopladas, en este método se requiere que las secuencias sean acopladas ya que cada secuencia representa los valores escalares en *X*, *Y*, y *Z*.

El índice de vértices se segmenta en *caras*, una cara es un *camino* entre los vértices (nodos), se detecta fácilmente cuando se *repite* el vértice inicial en una sub-secuencia. En el ejemplo de la figura 26, el índice de vértices define seis caras, la primera se comprende de v_0, v_1, v_2, v_3, y v_0; la segunda de v_2, v_6, v_7, v_3, y v_2; la tercera de v_6, v_5, v_4, v_7, y v_6; la cuarta de v_1, v_2, v_6, v_5, y v_1; la quinta de v_0, v_4, v_7, v_3, y v_0; y la última de v_1, v_5, v_4, v_0, y v_1.

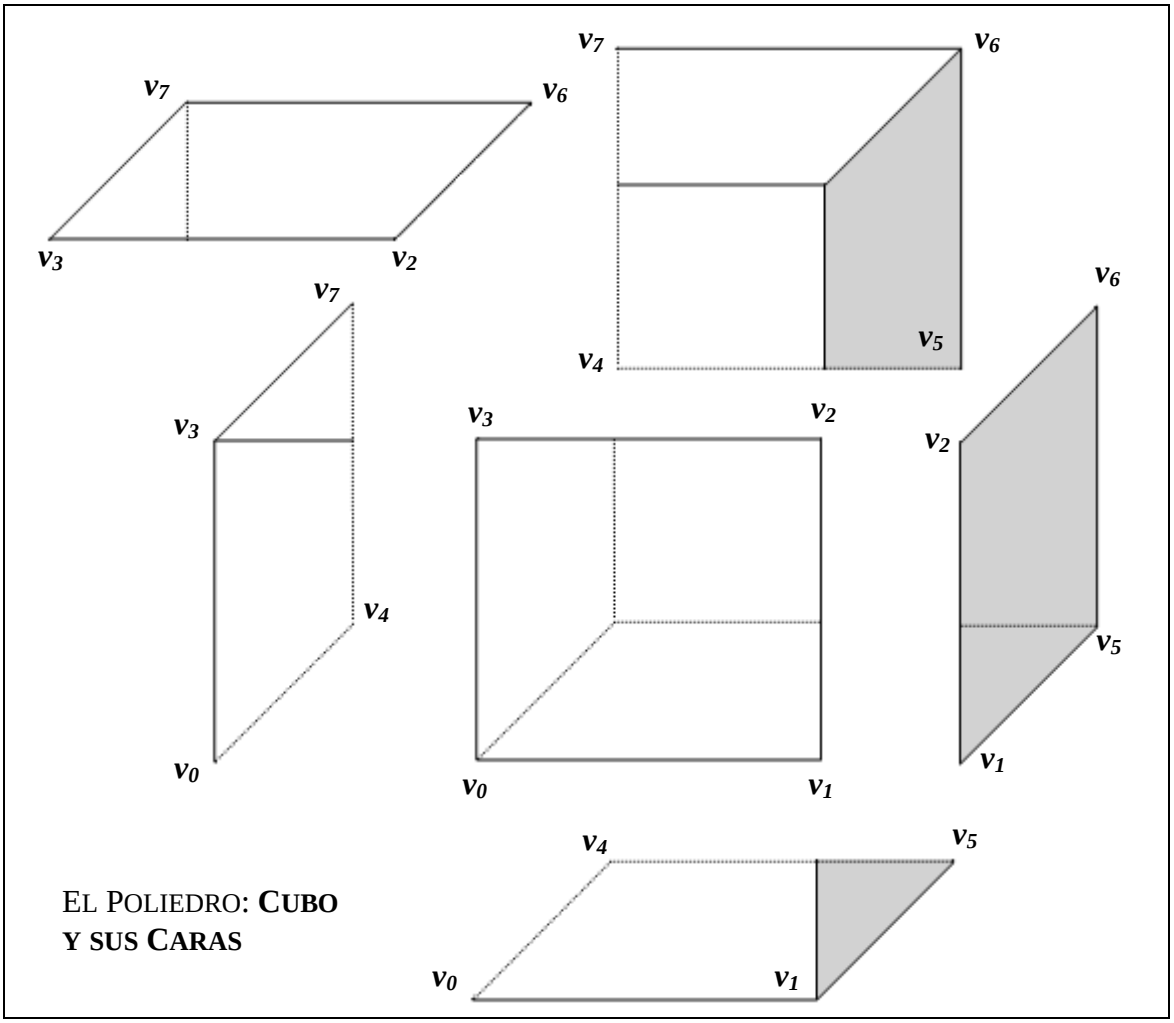


Figura 26. Caras en figuras geométricas. La segmentación del índice de vértices implica las caras del objeto o modelo 3D en cuestión.

Ahora que se ha detallado los aspectos más importantes del arreglo de vértices y el índice de vértices, a continuación se introduce la representación de figuras geométricas en una tripleta de secuencias acopladas.

El siguiente código en C++ que se ejecuta en *O*(n), donde n es la longitud del índice de vértices construye una matriz para almacenar las secuencias acopladas.

```
Objetivo: Representar figuras geométricas en secuencias acopladas
Parámetros: Arreglo de vértices V y índice de vértices F
Salida: Matriz M de 3x|F|, la primer columna es secuencia X, la
        segunda es la secuencia Y, y la tercera es la secuencia Z
```

```
for (int i=0; i < length(F); i++)
{
  M[0][i] = V[F[i]].x;     //iésimo elemento de la secuencia X
  M[1][i] = V[F[i]].y;     //iésimo elemento de la secuencia Y
  M[2][i] = V[F[i]].z;     //iésimo elemento de la secuencia Z
}
```

A continuación se muestra en la figura 27, la tripleta de secuencias acopladas, la connotación de *acoplada* se debe a que el iésimo elemento de las secuencias (X, Y, y Z), constituyen el iésimo vértice de la figura geométrica en cuestión.

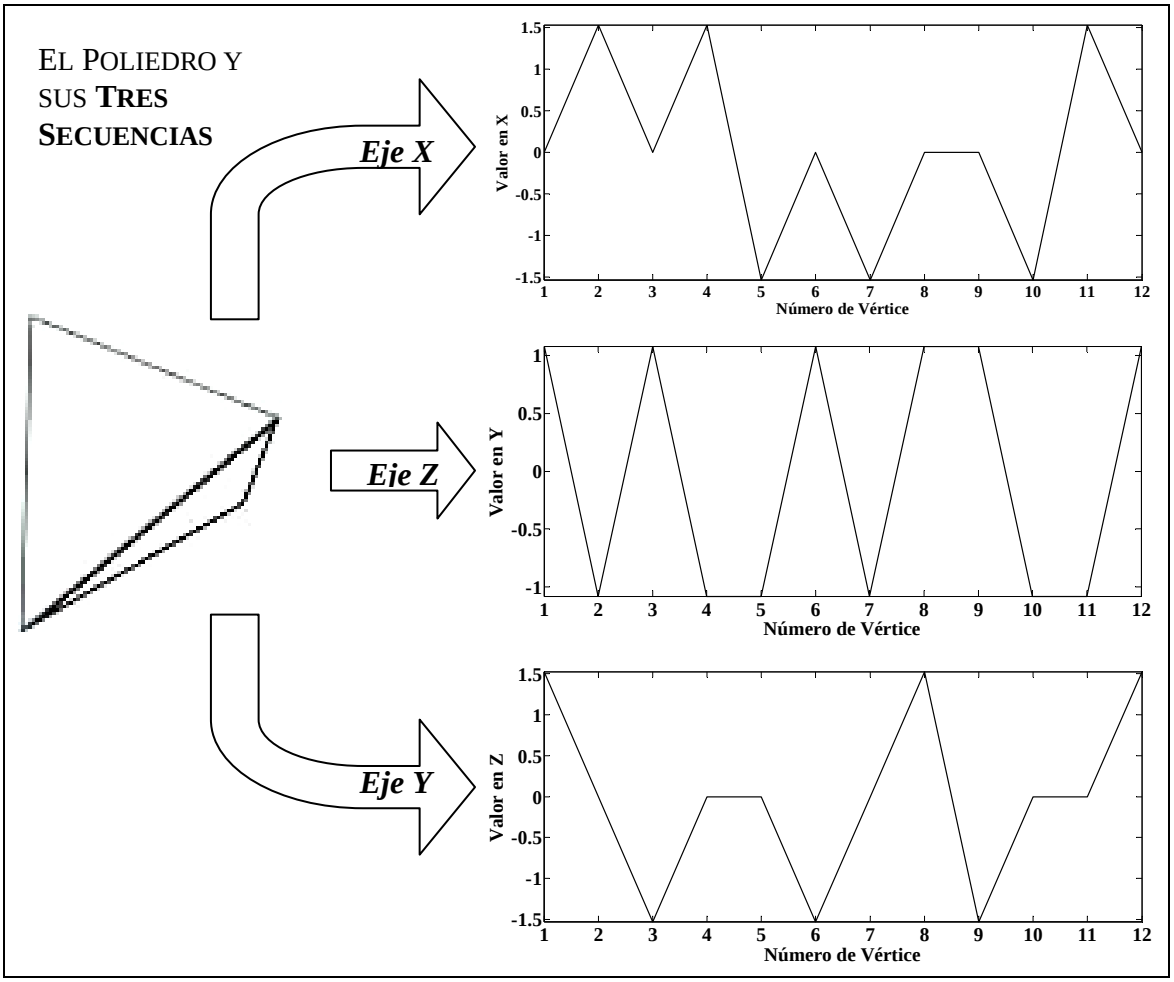


Figura 27. Representación de una figura en tres secuencias acopladas. Secuencias construidas a partir del arreglo de vértices e índices de éstos.

7.2 TRANSFORMACIONES BÁSICAS

Un aspecto importante que se persigue en los métodos de semejanza entre figuras geométricas, es la capacidad del *sistema* de reconocer un par de entidades distinguidas por una o más *transformaciones*.

Algunas de estas transformaciones son: *escala*, *traslación*, y *rotación*, estas son consideradas como trasformaciones *lineales* básicas. Otro tipo de transformaciones pueden ser creadas a partir de combinaciones de escalamiento, traslación, y rotación.

El lector podrá observar en secciones posteriores, que la presencia de dichas transformaciones no imposibilita al método de semejanza propuesto en este trabajo a reconocer un par de objetos que difieren en una o más transformaciones.

Lo anterior se logra cuando se representan entidades tridimensionales en una tripleta de secuencias, ya que ello permite usar técnicas para determinar semejanza entre series de tiempo, señales, secuencias, vectores, etc.

La escala, traslación, y rotación de una entidad tridimensional puede realizarse al multiplicar el *arreglo de vértices* y la *matriz adecuada*, esto indica que el índice de vértices **no es modificado**, y por lo tanto el orden de la secuencia es invariante a las transformaciones. A continuación se listan las transformaciones y matrices prototipo.

Tabla 1. Matrices prototipo y su uso para algunas trasformaciones lineales.

Trans-formación	***Matriz Prototipo***	***Uso***
Escala	$\begin{vmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$	$\begin{vmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} x_i \\ y_i \\ z_i \\ 1 \end{vmatrix} = \begin{vmatrix} x_i * x \\ y_i * y \\ z_i * z \\ 1 \end{vmatrix}$
Traslación	$\begin{vmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{vmatrix}$	$\begin{vmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} x_i \\ y_i \\ z_i \\ 1 \end{vmatrix} = \begin{vmatrix} x_i + x_i * x \\ y_i + y_i * y \\ z_i + z_i * z \\ 1 \end{vmatrix}$
Rotación sobre *x*	$\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(t) & -\sin(t) & 0 \\ 0 & \sin(t) & \cos(t) & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$	$\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(t) & -\sin(t) & 0 \\ 0 & \sin(t) & \cos(t) & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} x_i \\ y_i \\ z_i \\ 1 \end{vmatrix} = \begin{vmatrix} x_i \\ y_i * \cos(t) - y_i * \sin(t) \\ z_i * \sin(t) + z_i * \cos(t) \\ 1 \end{vmatrix}$
Rotación sobre *y*	$\begin{vmatrix} \cos(t) & 0 & \sin(t) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(t) & 0 & \cos(t) & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$	$\begin{vmatrix} \cos(t) & 0 & \sin(t) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(t) & 0 & \cos(t) & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} x_i \\ y_i \\ z_i \\ 1 \end{vmatrix} = \begin{vmatrix} x_i * \cos(t) + x_i * \sin(t) \\ y_i \\ z_i * -\sin(t) + z_i * \cos(t) \\ 1 \end{vmatrix}$
Rotación sobre *z*	$\begin{vmatrix} \cos(t) & -\sin(t) & 0 & 0 \\ \sin(t) & \cos(t) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$	$\begin{vmatrix} \cos(t) & -\sin(t) & 0 & 0 \\ \sin(t) & \cos(t) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} x_i \\ y_i \\ z_i \\ 1 \end{vmatrix} = \begin{vmatrix} x_i * \cos(t) - x_i * \sin(t) \\ y_i * \sin(t) + y_i * \cos(t) \\ z_i \\ 1 \end{vmatrix}$

Las matrices prototipo operan sobre el *arreglo de vértices* para llevar a cabo una transformación específica. Ello implica que el orden de los nodos o vértices no es modificado, sin importar el número de transformaciones que se le puedan realizar a una figura geométrica en particular.

En la siguiente figura se muestra un par de gráficas, la primera (de arriba a abajo), corresponde a la tripleta de secuencias perteneciente al cubo mostrado en la figura 24; la segunda gráfica, corresponde a la tripleta de secuencias del cubo de la figura 24, escalado 3 veces. Nótese que ambas gráficas comprenden de 30 puntos (longitud del índice de vértices), y siendo la discrepancia la *amplitud* de los valores para los vértices en *X*, *Y*, y *Z*.

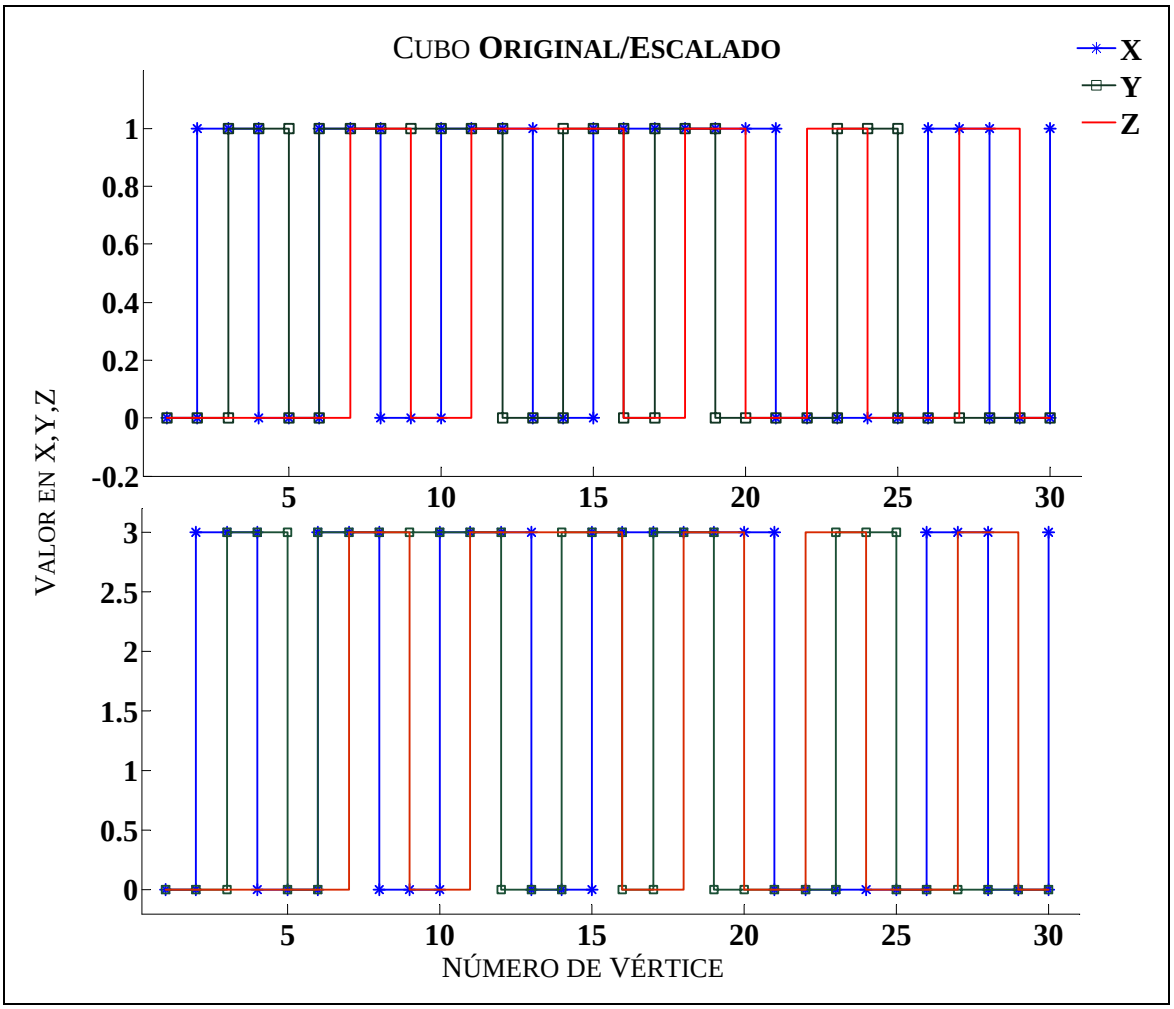


Figura 28. Escalamiento de un cubo. Nótese que la trasformación de escalamiento ha sido con un factor de 3, lo que produce una transformación de la tripleta de secuencias en amplitud.

Las matrices prototipo listadas en la tabla 1, son de 4x4, esto se debe a la incorporación de un *vértice ficticio*, denominado *w*, dicho vértice sólo es incorporado para facilitar las transformaciones. El lector podrá notar la conveniencia de este vértice ficticio en el cálculo de la translación, ya que sin este, se debería utilizar la matriz

prototipo de la trasformación escala, para poder llevar a cabo la misma operación. Por ejemplo, la transformación escalamiento se define como:

$$\begin{vmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} x_i \\ y_i \\ z_i \\ 1 \end{vmatrix} = \begin{vmatrix} x_i * x \\ y_i * y \\ z_i * z \\ 1 \end{vmatrix}$$

Es decir, por cada vértice del modelo 3D (para el cubo son 8), se deben determinar:

$x_i*x + x_i*0 + x_i*0 + x_i*0 \longrightarrow$ **Nuevo valor de** x_i

$y_i*0 + y_i*y + y_i*0 + y_i*0 \longrightarrow$ **Nuevo valor de** y_i

$z_i*0 + z_i*0 + z_i*z + z_i*0 \longrightarrow$ **Nuevo valor de** z_i

$1*0 + 1*0 + 1*0 + 1*1 \longrightarrow$ **1**

Donde x_i, y_i, y z_i, componen el *i-ésimo* vértice de la figura geométrica.

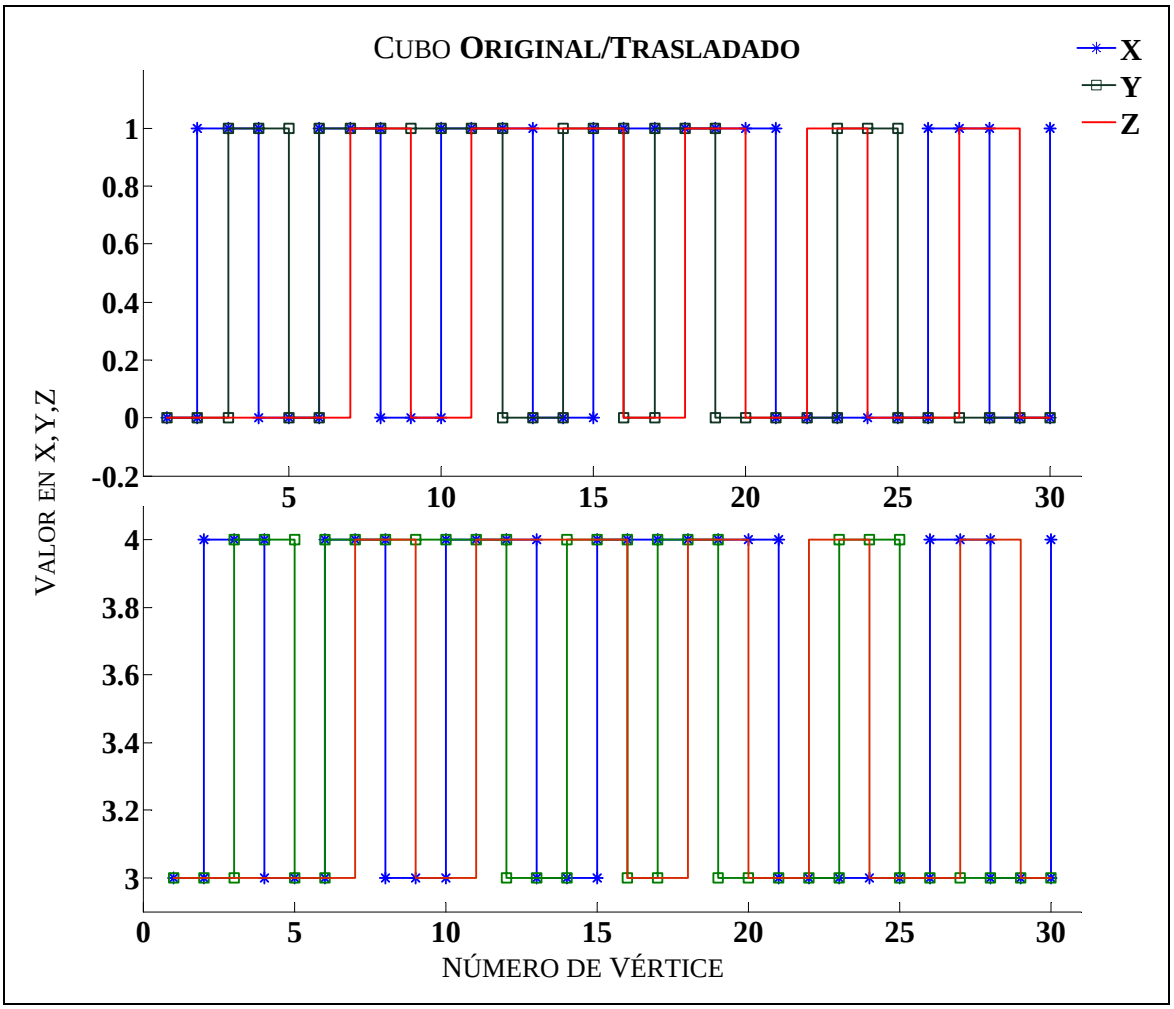


Figura 29. Traslación de un cubo. Nótese que la trasformación de traslación ha sido uniforme con un valor de 3, es decir, x=3, y=3, y z=3, para dicho caso es similar a la trasformación de escalamiento.

En la figura 29 se muestra un caso análogo a la figura 28, la diferencia radica en que la segunda gráfica (de arriba a abajo), ilustra la trasformación *traslación*. Nótese la sutil diferencia de las matrices prototipo de las transformaciones escalamiento y traslación (figura 28 y 29 respectivamente), y el resultado que éstas transformaciones logran sobre el mismo conjunto de secuencias, aún y cuando se aplican los mismos valores de parámetros en ambas matrices.

Para el lector familiarizado con el problema de semejanza en series de tiempo, señales, secuencias, arreglos, vectores, puede observar claramente el beneficio de abstraer las figuras geométricas como secuencias acopladas, en vez de usar algoritmos que busquen sub-grafos, aspectos, volúmenes, histogramas, vistas 2D, medidas heurísticas, etc.; tal y como se han planteado en la sección del *estado del arte*.

Gracias al gran avance que ha tenido el problema de semejanza en series de tiempo, es que se pueden proponer los métodos aquí introducidos. A continuación se muestran ejemplos de las transformaciones de rotación sobre un eje determinado, con un ángulo de 45°.

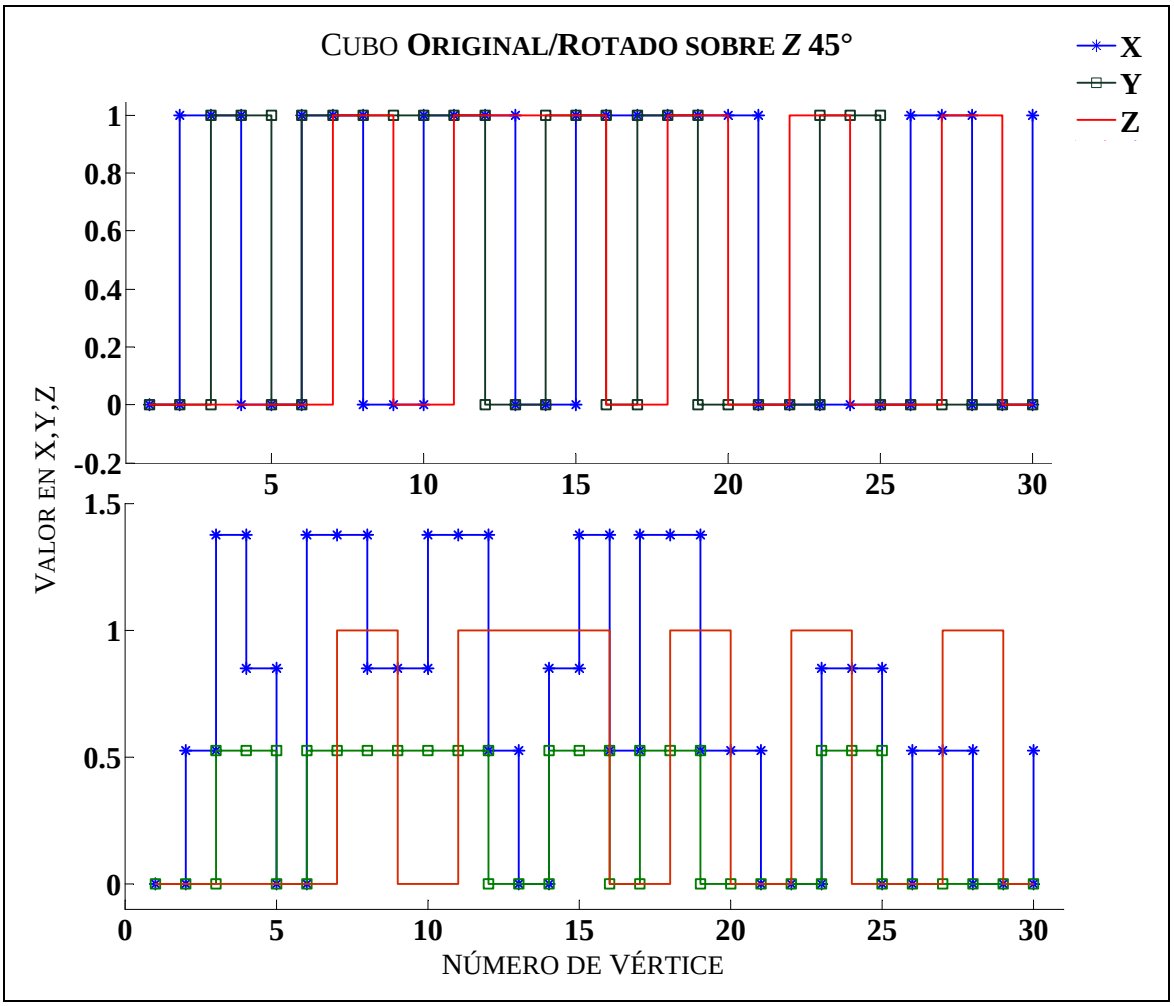


Figura 30. Rotación sobre *Z* de un cubo. Nótese que la trasformación de rotación sobre el eje *Z*, afecta los ejes *X* y *Y*, siendo invariante la secuencia *Z*. El ángulo de rotación es de 45°.

Cuando una transformación de rotación se aplica a una figura, aparentemente afecta en gran medida a la tripleta de secuencias.

Sin embargo, que la rotación sea estimada sobre un eje y ángulo específico, implica que dicho eje permanezca invariante, por ejemplo, la rotación del cubo de la figura 24, sobre el eje *Z*, con un ángulo de 45°, ver figura 30, se lleva a cabo de la siguiente forma:

$$\begin{vmatrix} \cos(t) & -\sin(t) & 0 & 0 \\ \sin(t) & \cos(t) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} x_i \\ y_i \\ z_i \\ 1 \end{vmatrix} = \begin{vmatrix} x_i * \cos(t) - x_i * \sin(t) \\ y_i * \sin(t) + y_i * \cos(t) \\ z_i \\ 1 \end{vmatrix}$$

Donde x_i, y_i, y z_i, son componentes en *X*, *Y*, y *Z*, respectivamente, del i-ésimo vértice de la figura geométrica. Para ver otras transformaciones ver el condensado de la tabla 1 en páginas anteriores.

A continuación se muestra en la figura Y, la rotación de un cubo sobre el eje Y.

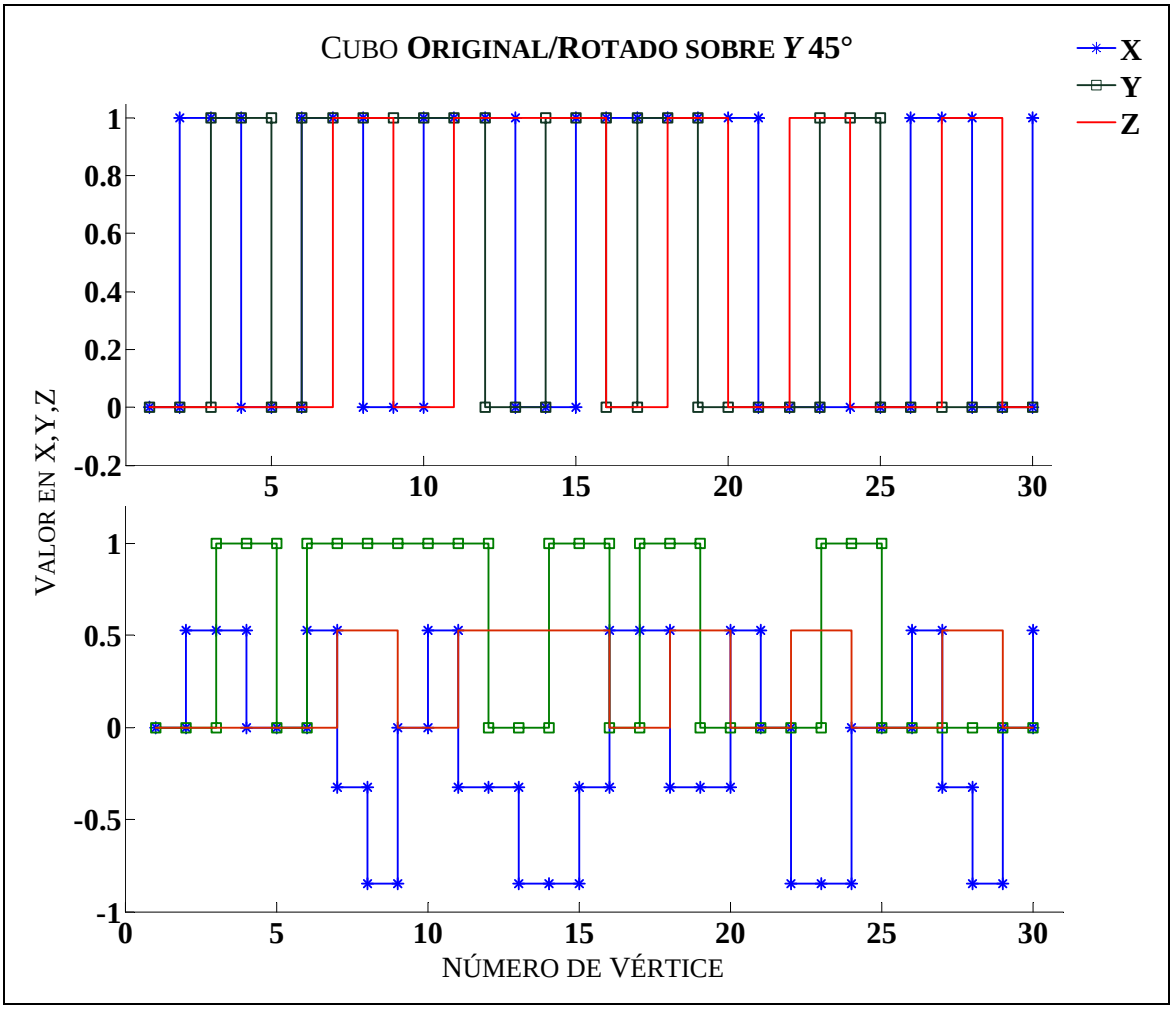


Figura 31. Rotación sobre *Y* de un cubo. Nótese que la trasformación de rotación sobre el eje *Y*, afecta los ejes *X* y *Z*, siendo invariante la secuencia *Y*. El ángulo de rotación es de 45°.

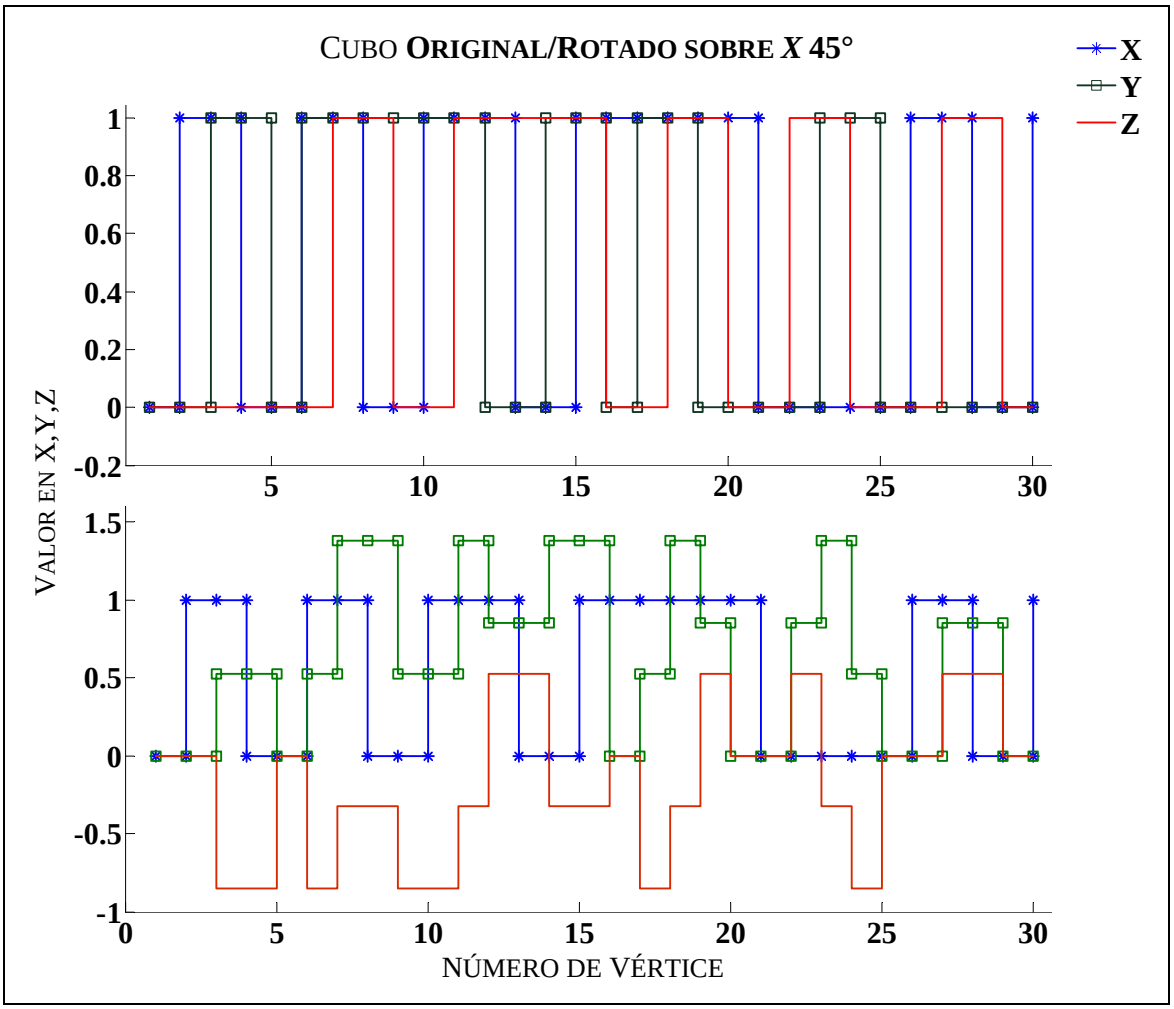


Figura 32. Rotación sobre *X* de un cubo. Nótese que la trasformación de rotación sobre el eje *X*, afecta los ejes *Y* y *Z*, siendo invariante la secuencia *X*. El ángulo de rotación es de 45°.

Para finalizar, en la figura 32, se muestra la transformación de rotación del cubo de la figura 24, sobre el eje *X*, con un ángulo de 45°. Nótese que la secuencia *X* permanece invariante a la transformación.

Cuando se observa sistemáticamente las afecciones que producen las transformaciones lineales sobre la representación propuesta en este capítulo (tripleta de secuencias), resulta *casi evidente* el siguiente paso, es decir, elaborar un método o estrategia de comparación que contemple dicha representación.

Las transformaciones de escalamiento y traslación representan el caso en que las afecciones se dan en menor o mayor medida en **todos** los ejes. Es decir, que sólo una secuencia (*X*, *Y* ó *Z*) basta para determinar una relación de semejanza entre dos entidades similares que sean diferenciadas únicamente por la escala o por la traslación.

Sin embargo, de las mismas observaciones y pruebas preliminares, se contempla un segundo caso; el de la rotación. En el caso de las rotaciones de objetos 3D, las

secuencias que permanecen invariantes (ejes de referencia), permiten argumentar la posibilidad de utilizar métodos de semejanza entre series de tiempo.

Al igual que en muchos otros problemas, el problema de representación se compone como un elemento destacado en la estrategia de comparación.

En la siguiente sección se presentará un método de comparación o semejanza que considera las ventajas de la representación de figuras geométricas aquí introducida.

7.3 SIMILITUD USANDO RNA'S FUZZY ART

En la sección anterior se mostró cómo representar entidades tridimensionales a través de secuencias acopladas.

Dicha representación presenta tres aspectos importantes:

1. La posibilidad de determinar una relación de semejanza por medio de un par de tripletas de secuencias, que corresponden a dos entidades tridimensionales.
2. La utilidad de incorporar métodos de semejanza para series de tiempo.
3. La capacidad de comparar figuras por uno, dos, o tres componentes; dichos componentes corresponden a las coordenadas en *X*, *Y*, y *Z*.

El objetivo de cualquier estrategia de representación es facilitar la resolución de un problema específico, en particular las figuras geométricas de este trabajo, sin embargo existen otros problemas que pueden ser abstraídos de forma similar.

Como ya se ha mostrado, la tripleta de secuencias acopladas representa la geometría de la figura en cuestión. La tarea por resolver es la de determinar una relación de semejanza entre figuras a través de sus respectivas secuencias acopladas; esto es, el problema de determinar semejanza entre secuencias, series de tiempo, señales, vectores. Debido a que se ha considerado una tripleta de secuencias, se formula la siguiente regla:

"Comparar elementos comparables"

Comparar secuencias comparables, es decir, establecer una relación de semejanza entre figuras geométricas por medio de sus respectivas secuencias en *X*, *Y*, y *Z*.

En el capítulo 4 se describió la arquitectura de la *RNA Fuzzy ART*, esta arquitectura clasifica patrones de entrada (en forma de secuencias), lo cual es exactamente lo que se desea hacer con la tripleta de secuencias acopladas.

Para ejemplificar lo anterior supóngase que se consideran tres figuras, *poliedro03*, *poliedro04*, y *piramide02*, como se muestra en la figura 33; una tripleta de secuencias acopladas es creada por cada figura geométrica, generando tres secuencias *X*, tres secuencias *Y*, y tres secuencias *Z*, en la figura 33 se muestran dichas figuras y la correspondiente gráfica de la tripleta de secuencias acopladas.

Las dos primeras figuras son iguales, uno está rotado 180° con respecto al otro, referente al eje *Y*, las secuencias de dichas figuras constan de 12 puntos, es decir, el índice de vértices tanto del *poliedro03* y *poliedro04* es de longitud 12, la figura *pirámide02* aunque es similar a las dos primeras figuras, resulta ser una figura geométrica más complicada, las respectivas secuencias constan 25 puntos.

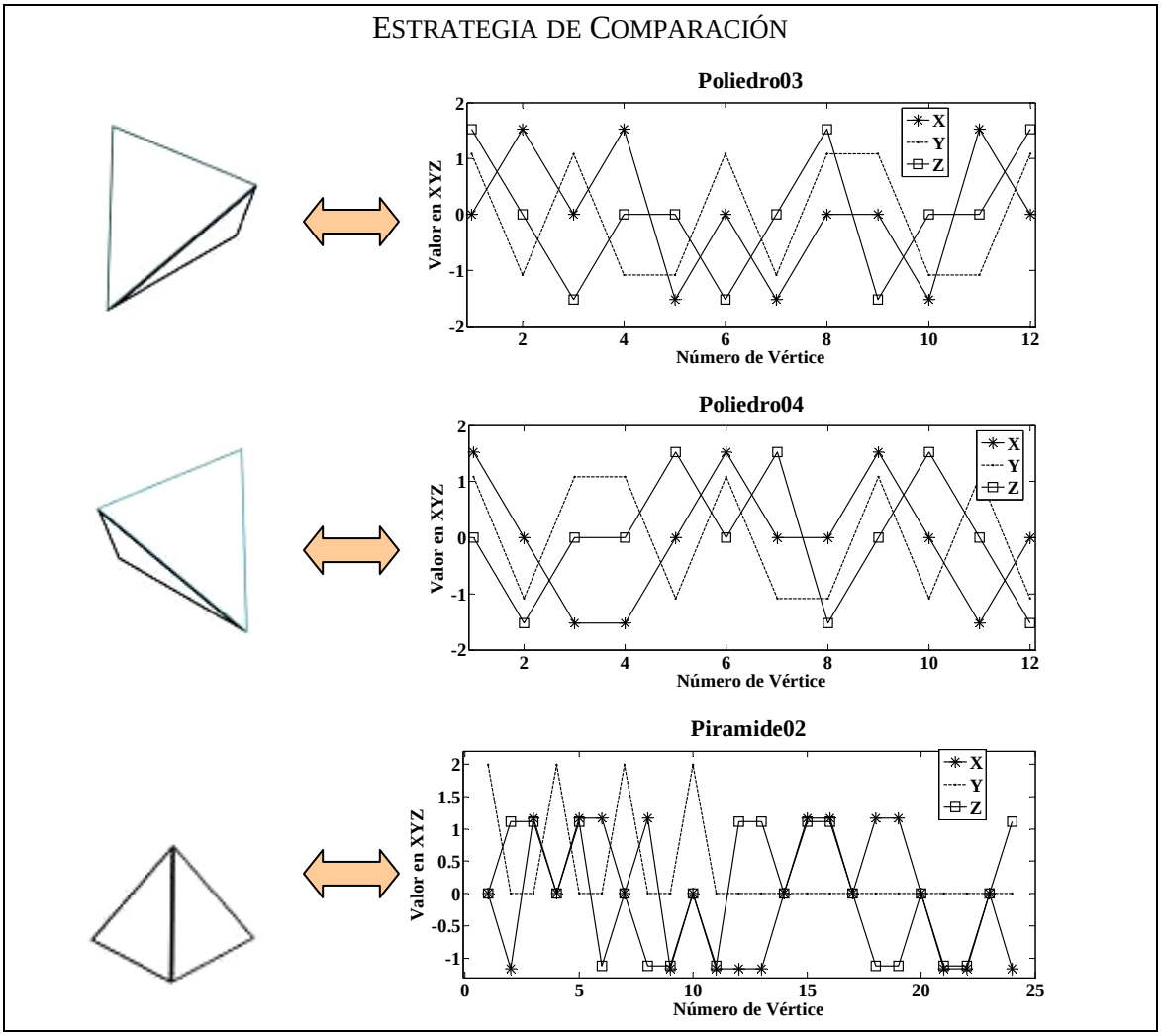


Figura 33. Estrategia de comparación de figuras geométricas. Poliedro03, Poliedro04, Piramide02 con sus respectivas secuencias acopladas.

Aunque en la figura 33 puede ser observarse la importancia de comparar secuencias de un mismo eje, resulta más claro en la figura 34 la comparación por coordenadas, es decir, la primera gráfica (de arriba a abajo), ilustra las secuencias de las figuras en el eje

X, la segunda gráfica comprende de secuencias sólo de eje *Y*; por último la tercera gráfica comprende sólo de secuencias que corresponden al eje *Z*.

El lector podrá observar que las secuencias de las figuras *poliedro03* y *poliedro04* en la coordenada *X* y *Z*, son similares, una invertida con respecto a la otra; en el eje *Y*, éstas dos figuras geométricas son casi idénticas. Como se explicó en el capítulo 4, las propiedades de la *RNA Fuzzy ART*, son convenientes para la detección de sub-secuencias (set/superset), así como la inversión de secuencias.

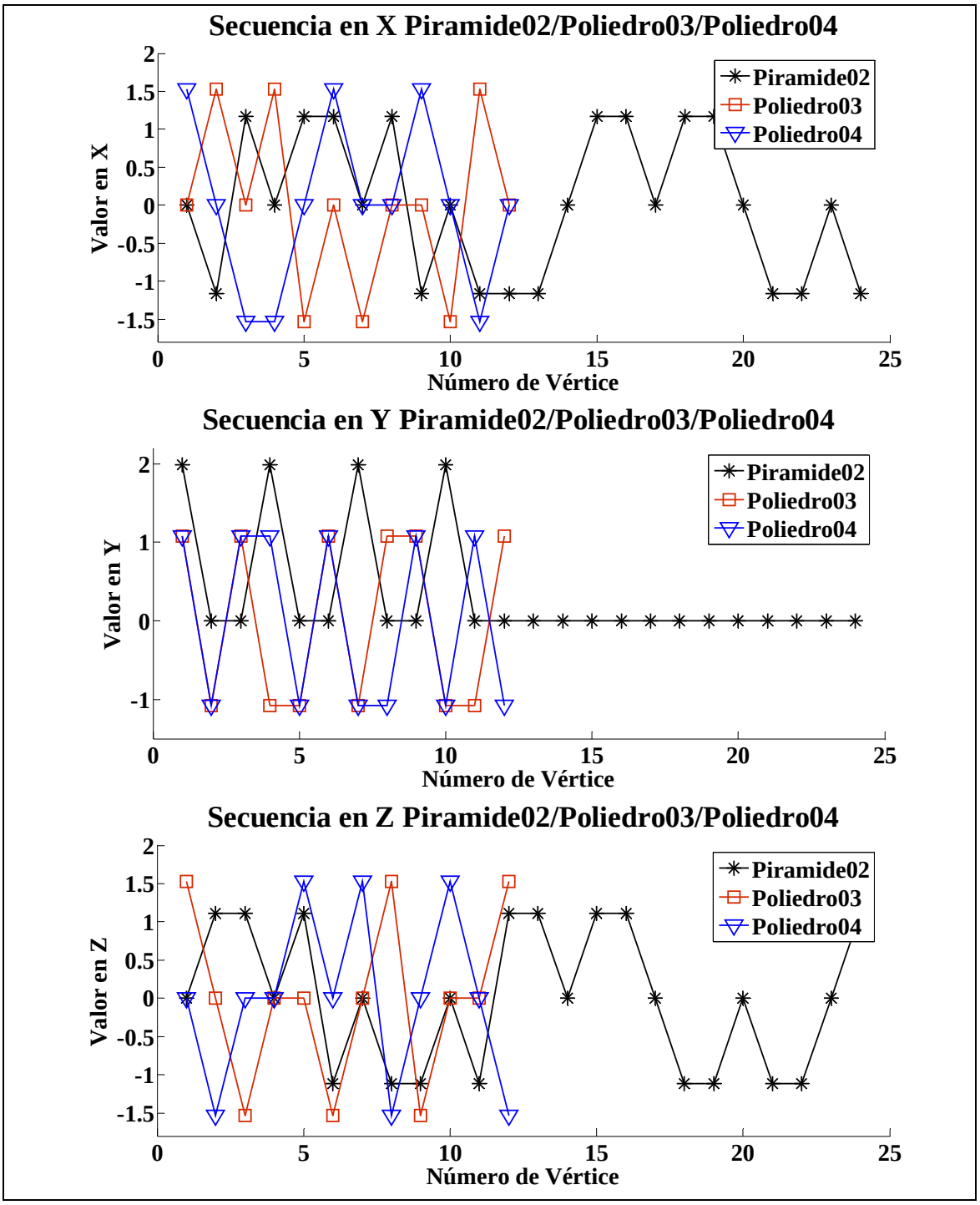


Figura 34. Comparación de figuras geométricas a través de sus secuencias acopladas.

La existencia de tres secuencias por cada figura geométrica, sugiere la posibilidad de implementar **tres *RNA's Fuzzy ART***. Cada clasificador (ART_X, ART_Y, ART_Z), podrá asignar una clase específica por coordenada a cada figura dentro de la base de datos, ver figura 35.

La tres *RNA Fuzzy ART*, deberán compartir los valores de parámetros, es decir, el número de clases, la longitud de las secuencias (constante), el parámetro de vigilancia *rho*, *beta* y *alfa*. Detalles de la arquitectura Fuzzy ART se encuentran en el capítulo 4.

La necesidad de que las secuencias sean **acopladas** surge en este momento. Los clasificadores serán alimentados en un orden específico a priori. Y cada clasificador sólo recibirá patrones de entrada de una coordenada específica.

Como se ha especificado en capítulo 6, las figuras utilizadas en este trabajo son básicamente de dos tipos: a) de tamaño constante, es decir, la longitud del índice de vértices es constante para un conjunto de datos específico, o b) de tamaño variable, que implica que la longitud del índice de vértices puede cambiar para cada par de figuras en una base de datos específica.

Como lo muestran los resultados experimentales en el capítulo 9, el método de semejanza entre figuras basado en *RNA's Fuzzy ART*, es idóneo para el primer tipo de figuras geométricas, es decir, figuras de tamaño constante.

Los resultados de estas pruebas experimentales reportadas en [Angeles05a] establecieron las bases para proponer un método complementario (idóneo para comparar figuras de tamaño variable), éste método se basa en la técnica *Dynamic Time Warping*, detalles sobre éste se encuentran en el capítulo 8.

Adicionalmente la implementación de tres *RNA's Fuzzy ART*, responde a la idea de comparar secuencias, que sean comparables, es decir, intuitivamente y experimentalmente puede inferirse que comparar secuencias de coordenadas distintas puede conducir a un error de apreciación geométrica.

El lector puede identificar que utilizar sólo dos secuencias (*X* y *Y*, ó *X* y *Z*, ó *Y* y *Z*), esto es, una por coordenada, resultaría en el paradigma de *aspectos* (ver el capítulo del estado del arte). Ya que sólo interesaría una imagen o vista de un objeto tridimensional.

Las *RNA's Fuzzy ART* han sido reconocidas por su poder de cómputo en el área de Redes Neuronales Artificiales, su eficacia y costo computacional las hacen idóneas para la tarea designada por el método de comparación. A continuación, en la figura 35 se ilustra el método de semejanza basado en *RNA's Fuzzy ART* propuesto en este capítulo.

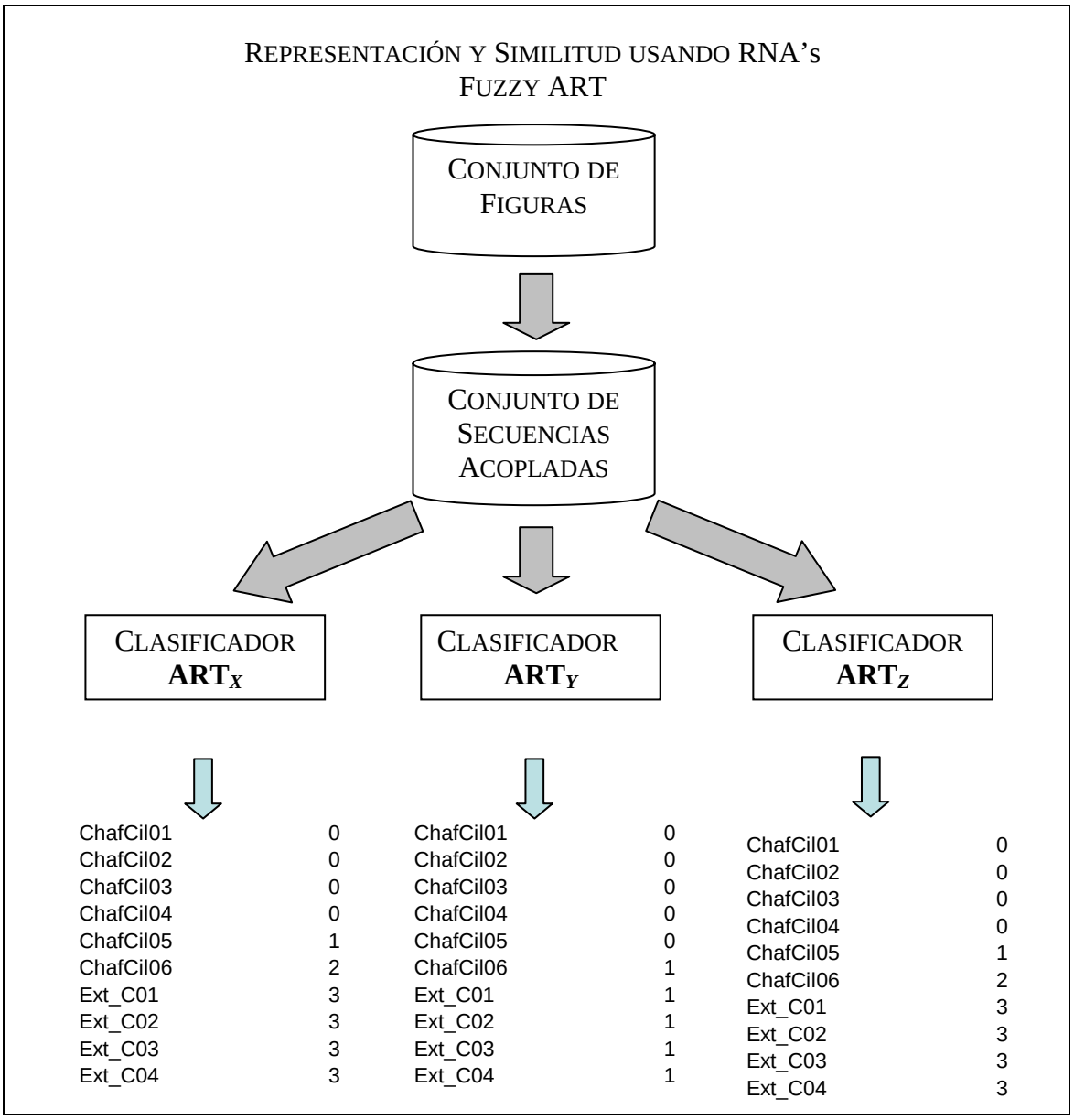


Figura 35. El modelo de búsqueda de semejanza entre objetos 3D por indexado.

En la figura 35, cada clasificador determina un valor entero, éste representa la clase a la que pertenece dicha figura considerando una sola coordenada. Como se muestra en la figura 35, el conjunto de secuencias acopladas deben ser alimentadas a los clasificadores de acuerdo a un orden preestablecido. Dicho orden puede ser por nombre de objeto, el cual tienen asociado una tripleta de secuencias acopladas.

El siguiente paso que se ha determinado es la **unificación de categorías**, dicha unificación surge de la observación sistemática de las clases que determinan los tres clasificadores (ART_X, ART_Y, ART_Z).

En este momento, podría especularse sobre la posibilidad de utilizar una solo coordenada, sin embargo, en el capítulo siguiente se muestra cómo convertir una tripleta de secuencias acopladas a una sola **secuencia representativa**.

La unificación de clases se realiza por medio de la distancia de Hamming, la distancia de Hamming ha sido utilizada para la detección y corrección de errores en transmisiones electrónicas.

La distancia de Hamming indica el número de bits en los que difieren dos cadenas binarias. Los clasificadores pueden ser abstraídos como emisores de cadenas binarias, la tripleta de categorías asignada a un objeto dado son convertidas a cadenas binarias. Por ejemplo, en la figura 35, se muestra la clasificación para una figura denominada ChafCil01, de acuerdo al ejemplo, la figura pertenece a la clase 0 considerando la coordenada *X*, a la clase 0 considerando la coordenada *Y*, y la clase 0, considerando la clase *Z*, nótese que no siempre las categorías por coordenadas serán congruentes, ver el ejemplo en la figura 35 del objeto Ext_01. A continuación se muestra la cadena binaria correspondiente a la tripleta de categorías asignada por los clasificadores.

ChafCil01:000000000000000000000000

Figura 36. Tripleta de categorías. Codificación binaria para la tripleta de categorías asignadas por los clasificadores ART_X, ART_Y, y ART_Z.

Una vez que las tripletas de categorías son convertidas a cadenas binarias, se busca la unificación de las clases, es decir, se desea determinar la una sola categoría por figura geométrica.

Por simplicidad en la figura 37 se muestra una lista de figuras geométricas y sus respectivas cadenas binarias (que representan 3 categorías en *X*, *Y*, y *Z*). La unificación se logra comparando la distancia de Hamming del primer elemento en la lista, contra el resto de los elementos en la lista, incluyendo él mismo.

El lector puede identificar que conforme aumente el tamaño de la lista la probabilidad de que la distancia de Hamming con el resto de los elementos en la lista cada vez sea mayor. Sin embargo ello no es determinante para conformar un criterio de paro anticipado al final de la lista.

En [Angeles05a] se sugiere establecer un criterio de corte, dicho criterio sirve para conformar o asignar una nueva clase definitiva, el valor de corte de define como la distancia de Hamming menor o igual a 1.

Por ejemplo, en la figura 37, el primer elemento en la lista es ChafCil01, dicho elemento calculará la distancia de Hamming (con sus respectivas cadenas binarias) contra ChafCil02, ChafCil03, ChafCil04, y ChafCil05, dado que el único elemento que excede el criterio de corte es ChafCil05, los 4 primeros elementos de la lista conforman la clase 1. En el caso de que la lista continuara el elemento ChafCil05 sería ahora la referencia, y procedería a computar la distancia de Hamming con el resto de la lista en

búsqueda de definir otro posible conjunto en el cual la distancia de Hamming no se exceda en 1.

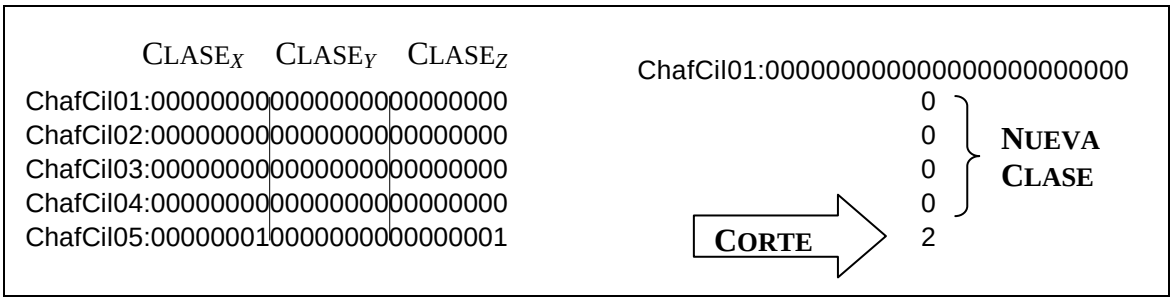


Figura 37. Unificación de clases.

Finalmente, la clasificación que se logra con el método propuesto en este capítulo se basa en la similitud que *RNA Fuzzy ART* detecta en los patrones de entrada, la representación de las figuras facilita la incorporación de dicha arquitectura; el cálculo de la distancia de Hamming surgió como una regla basada en la experiencia.

Temas derivados de la idea aquí introducida alcanzan a ser dilucidados en la sección de discusión y conclusiones.

El método de semejanza entre figuras geométricas usando *RNA Fuzzy ART* comprende de tres grandes etapas:

1. Representar las figuras geométricas en tripletas de secuencias acopladas.
2. Clasificar las tripletas de secuencias acopladas en *X*, *Y*, y *Z*.
3. Unificar la tripleta de categorías.

Resultados experimentales aplicando el método introducido en este capitulo se encuentran en la sección de Resultados, de igual forma, estos resultados han sido publicados en [Angeles05a].

REFERENCIAS DEL CAPÍTULO

[Angeles05a] A. Angeles-Yreta, J. Figueroa-Nazuno. "Semejanza entre Objetos Tridimensionales aplicando Redes Neuronales Fuzzy ART". *Advances in Computer Science in Mexico, Research on Computer Science*. Vol. 13, A. Gelbukh y H. Calvo (Eds.), ISSN: 1665-9899, 2005, pp. 127-134.

[Angeles05b] A. Angeles-Yreta, J. Figueroa-Nazuno. "Similarity Search in 3D Objects and Dynamic Time Warping". *Proceedings of International Seminar on Computational Intelligence*. IEEE CIS Mexico Chapter. 2005.

[Angeles05c] A. Angeles-Yreta, J. Figueroa-Nazuno. "Búsqueda de Semejanza entre Objetos Tridimensionales por Indexado". *Congreso Nacional de Física 2005*. ISSN 0187-4713. 2005. pp. 97

[Angeles05d] A. Angeles-Yreta, J. Figueroa-Nazuno. "Computing Similarity Among 3D Objects Using Dynamic Time Warping". *Lecture Notes in Computer Science* (Springer-Verlag), Progress in Pattern Recognition, Image Analysis and Applications: *10th Iberoamerican Congress on Pattern Recognition, CIARP 2005*, Havana, Cuba, November 15-18, 2005, ISSN: 0302-9743, ISSN: 0302-9743. pp. 319-326.

[Angeles05e] A. Angeles-Yreta, J. Figueroa-Nazuno, K. Ramírez-Amaro. "Búsqueda de Semejanza entre Objetos 3D por Indexado". *Decimosexta Reunión de Otoño Comunicaciones, Computación* ROC&C 2005. 2005.

[Segal04] Segal M., Akeley K. "The OpenGL Graphics System: A Specification". Leech J., Brown P. (Eds.). *Silicon Graphics Press*. 2004.

8 REPRESENTACIÓN Y CÓMPUTO DE SIMILITUD USANDO DTW

En este capítulo se introduce un método de cómputo de similitud que utiliza una secuencia para la comparación de entidades tridimensionales, dicho método puede verse como una evolución de método propuesto en capítulo 7, ya que los resultados experimentales obtenidos por éste dieron lugar a la creación de un método de comparación de entidades tridimensionales de tamaño variable.

8.1 REPRESENTACIÓN DE FIGURAS EN UNA SECUENCIA

En el capítulo anterior, se describió un método de cómputo de similitud entre figuras geométricas usando una representación directa, es decir, sin ningún tipo de extracción, selección y/o construcción de características clásicas, dicha representación comprendía en una tripleta de secuencias acopladas por figura geométrica.

Una etapa de unificación de clases fue requerida con el objeto de determinar grupos que sostienen una relación disimilitud. La relación de similitud fue establecida con la ayuda de tres clasificadores *RNA Fuzzy ART*, los detalles de esta arquitectura se encuentran en el capítulo 4.

Una de las grandes limitantes de la arquitectura *RNA Fuzzy ART*, es la incapacidad de clasificar patrones de entrada de tamaño variable, esto es, secuencias, series de tiempo, señales, etc., de longitud variable.

Las pruebas experimentales del método basado en *RNA Fuzzy ART*, mostraron que las categorías que asignan los tres clasificadores, no difieren en gran medida, es decir, incorporar tres secuencias que corresponden a cada eje cartesiano (asociado a una sola figura), no implica tener criterios distintos para la clasificación de una entidad tridimensional [Angeles05a].

De lo anterior surgió la idea de crear otra forma de representar las entidades tridimensionales, que fuera aún más eficaz y eficiente, esto es, que permita, clasificar

dichas entidades, con una solo secuencia representativa (en vez de tres secuencias acopladas), y que además complemente el método introducido en el capítulo 7.

Por comodidad se presenta nuevamente la figura 37, dicha figura muestra la congruencia de los tres clasificadores con respecto a la clase a la que debe pertenecer los primeros cuatro elementos de la lista independientemente de la secuencia (*X*, *Y*, y *Z*).

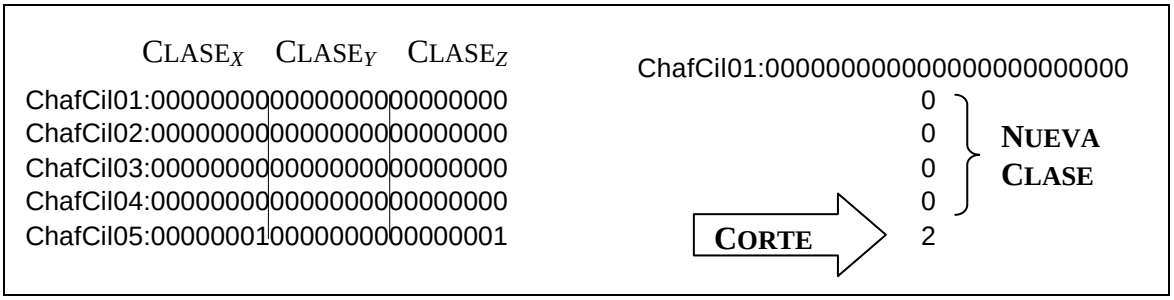


Figura 37. Unificación de clases.

Lo que se sugiere en [Angeles05b] [Angeles05c] [Angeles05d] [Angeles05e], es la re-etiquetado de los vértices que componen la figura en cuestión.

Dicho re-etiquetado consiste en la creación de una secuencia producto de la suma de las tres secuencias acopladas de una figura dada. En otras palabras, sea (**14**) y (**15**), el arreglo de vértice e índice de vértice:

$$V = \langle v_1, v_2, \ldots, v_{|V|} \rangle, \textit{ donde } v_i = (x_i, y_i, z_i) \; y \; x_i, y_i, z_i \in \Re \qquad (\mathbf{14})$$

$$F = \langle v_{f_1}, v_{f_2}, \ldots, v_{f_n} \rangle, \textit{ donde } f_j \in [1, |V|] \qquad (\mathbf{15})$$

El re-etiquetado se define como:

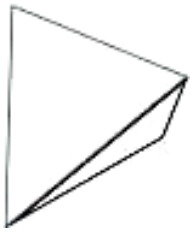
$$\begin{aligned} S &= \langle sv_{f_1}, sv_{f_2}, \ldots, sv_{f_n} \rangle, \textit{ donde} \\ sv_{f_j} &= (x_j + y_j + z_j) \; y \; f_j \in [1, |V|] \end{aligned} \qquad (\mathbf{16})$$

La idea latente es sustituir el índice de vértices con la suma de componentes en *X*, *Y*, y *Z*. Decimos que se crea una secuencia representativa por dos razones:

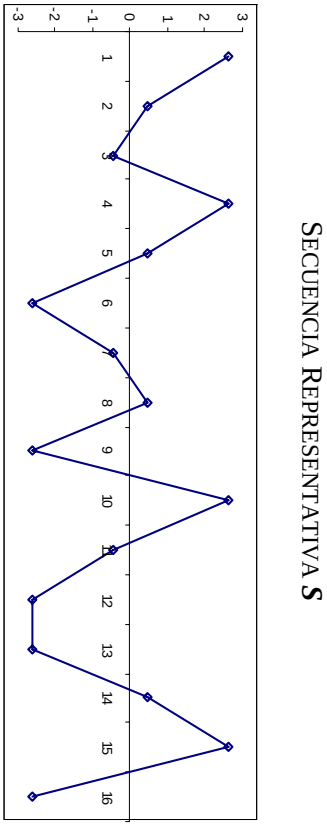
1) Nos indica el **orden** de los nodos.
2) Nos indica la **magnitud** de los nodos, la representación de (**14**) y (**15**) se codifica en una secuencia representativa.

A continuación se muestra En la figura 38 se muestra la secuencia representativa de una figura geométrica.

LA FIGURA Y SU **SECUENCIA REPRESENTATIVA**



VÉRTICES	***S***
$v_1 = (0,1.081,1.529)$	**2.61**
$v_2 = (1.529,-1.081,0)$	**0.448**
$v_0 = (0,1.081,-1.529)$	**-0.45**
$v_1 = (0,1.081,1.529)$	**2.61**
$v_2 = (1.529,-1.081,0)$	**0.448**
$v_3 = (-1.529,-1.081,0)$	**-2.61**
$v_0 = (0,1.081,-1.529)$	**-0.45**
$v_2 = (1.529,-1.081,0)$	**0.448**
$v_3 = (-1.529,-1.081,0)$	**-2.61**
$v_1 = (0,1.081,1.529)$	**2.61**
$v_0 = (0,1.081,-1.529)$	**-0.45**
$v_3 = (-1.529,-1.081,0)$	**-2.61**
$v_3 = (-1.529,-1.081,0)$	**-2.61**
$v_2 = (1.529,-1.081,0)$	**0.448**
$v_1 = (0,1.081,1.529)$	**2.61**
$v_3 = (-1.529,-1.081,0)$	**-2.61**



SECUENCIA REPRESENTATIVA ***S***

Figura 38. Representación una figura geométrica en una sola secuencia representativa. Secuencia construida a partir del arreglo de vértices e índice de éstos.

En la figura 38, se ilustra uno de los objetos usados en las figuras del capítulo anterior. El *poliedro03*, se define como un objeto de 4 vértices, y cuatro caras. La secuencia representativa resultante de la suma de los componentes x_i, y_i, y z_i, del vértice i, crea una secuencia de 16 puntos.

Como ya se ha mencionado en párrafos anteriores, dicha secuencia *representa*, el orden de los vértices de la entidad tridimensional, así como la magnitud de dicho vértice; la magnitud es forzada a una sola dirección.

Otra ventaja destacable, es la posibilidad de compara figuras geométricas (por medio de su respectiva secuencia), usando *DTW* o *FDTW*, detalles sobre estas técnicas se encuentran en capítulo 5 del presente trabajo.

Para ejemplificar lo anterior se muestra en la figura 39, un ejemplo similar al expuesto en la figura 33 del capítulo 7. En dicha figura se muestran tres objetos,

poliedro03, *poliedro04*, y *piramide02*, con su respectiva secuencia representativa, la longitud de estas secuencias ha sido recortada por simplicidad.

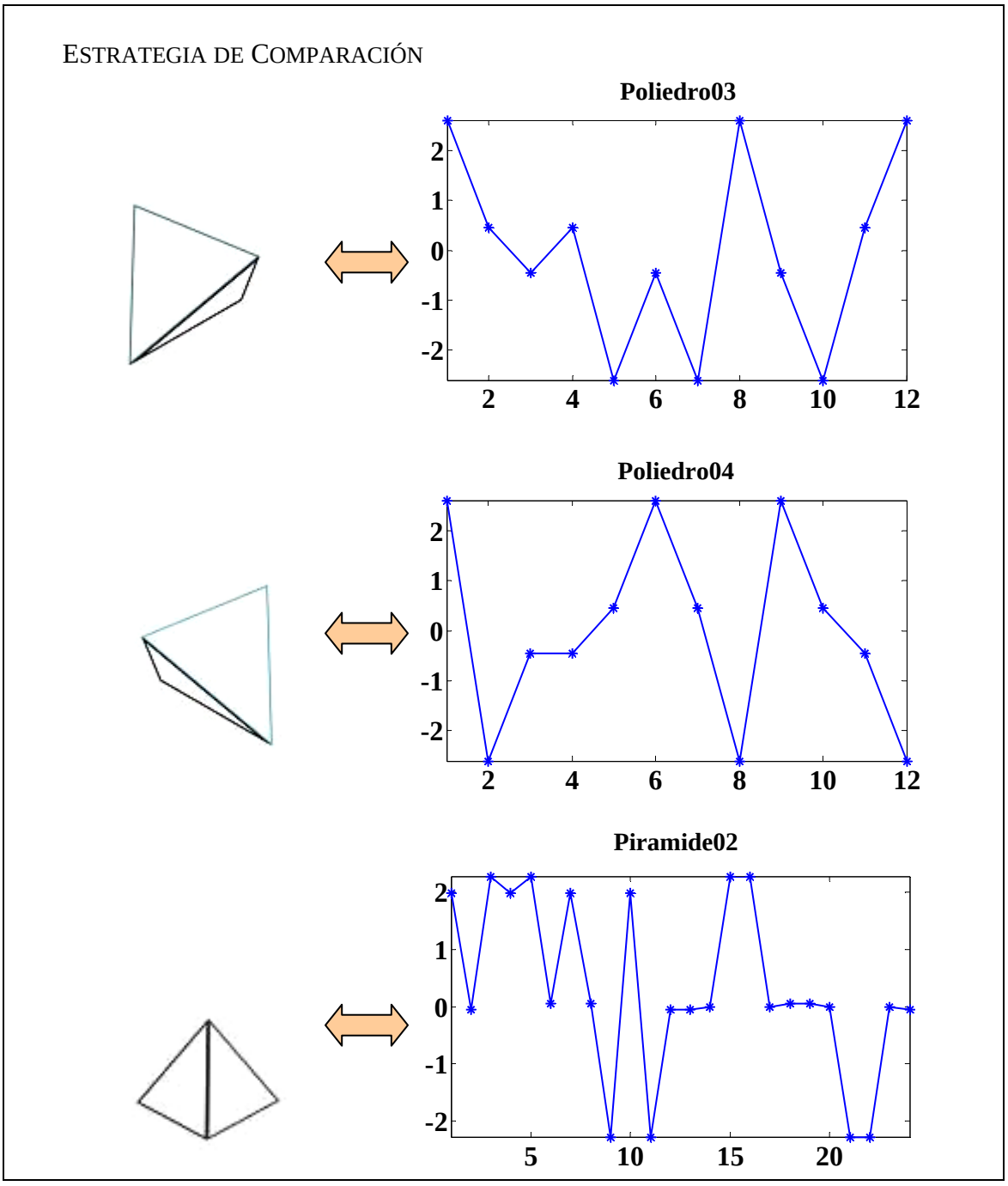


Figura 39. Estrategia de comparación de figuras geométricas. *Poliedro03*, *Poliedro04*, *Piramide02* con sus respectivas secuencias representativas.

De forma análoga al método de similitud del capítulo anterior, el problema de calcular similitud entre figuras geométricas ha sido reducido al problema de computar similitud entre secuencias, series de tiempo, señales, arreglos, etc.

En la figura 40, se ilustra el problema de comprar los objetos: *poliedro03*, *poliedro04*, y *piramide02*.

Para evitar la reducción de dimensionalidad se ha empleado un algoritmo variante de *DTW*, cuya complejidad permite realizar una búsqueda secuencial en tiempo polinomial.

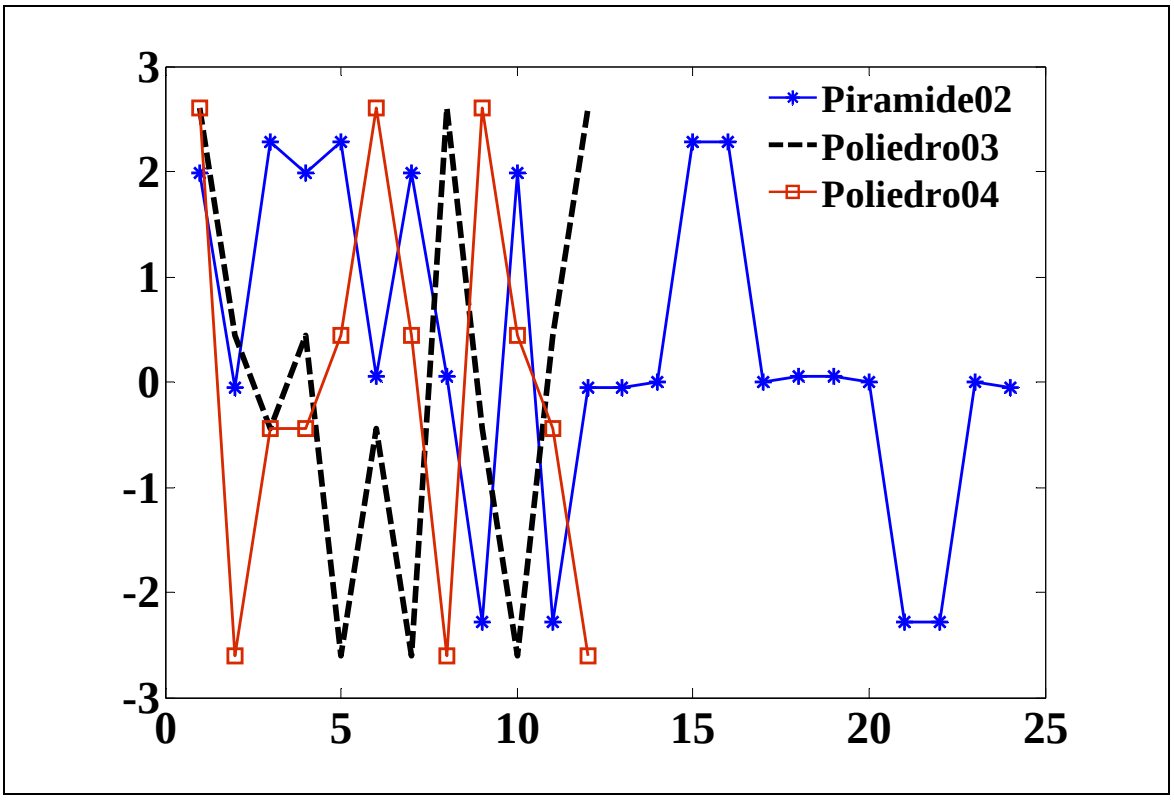


Figura 40. Comparación de figuras geométricas a través de secuencias representativas.

8.2 CÓMPUTO DE SIMILITUD USANDO DTW

DTW ha sido aplicado principalmente para el reconocimiento de voz [Deller00], la complejidad del algoritmo original [Kim01], no ha permitido emplearlo para secuencias medianamente grandes.

Sin embargo, la implementación de Jang en [Jang00], permite realizar en tiempo y espacio polinomial el cálculo de la distancia *DTW*, entre dos secuencias dadas.

Desde una perspectiva simplista, el éxito del método de similitud propuesto en este capítulo, radica en la adecuada representación del problema, esto es, compactando una estructura tridimensional a una estructura unidimensional, conservando a la vez, los elementos necesarios para caracterizar suficientemente un par de figuras geométricas para su distinción.

A continuación se ilustra el método para computar similitud entre figuras, usando secuencias representativas y *DTW* o *FDTW*.

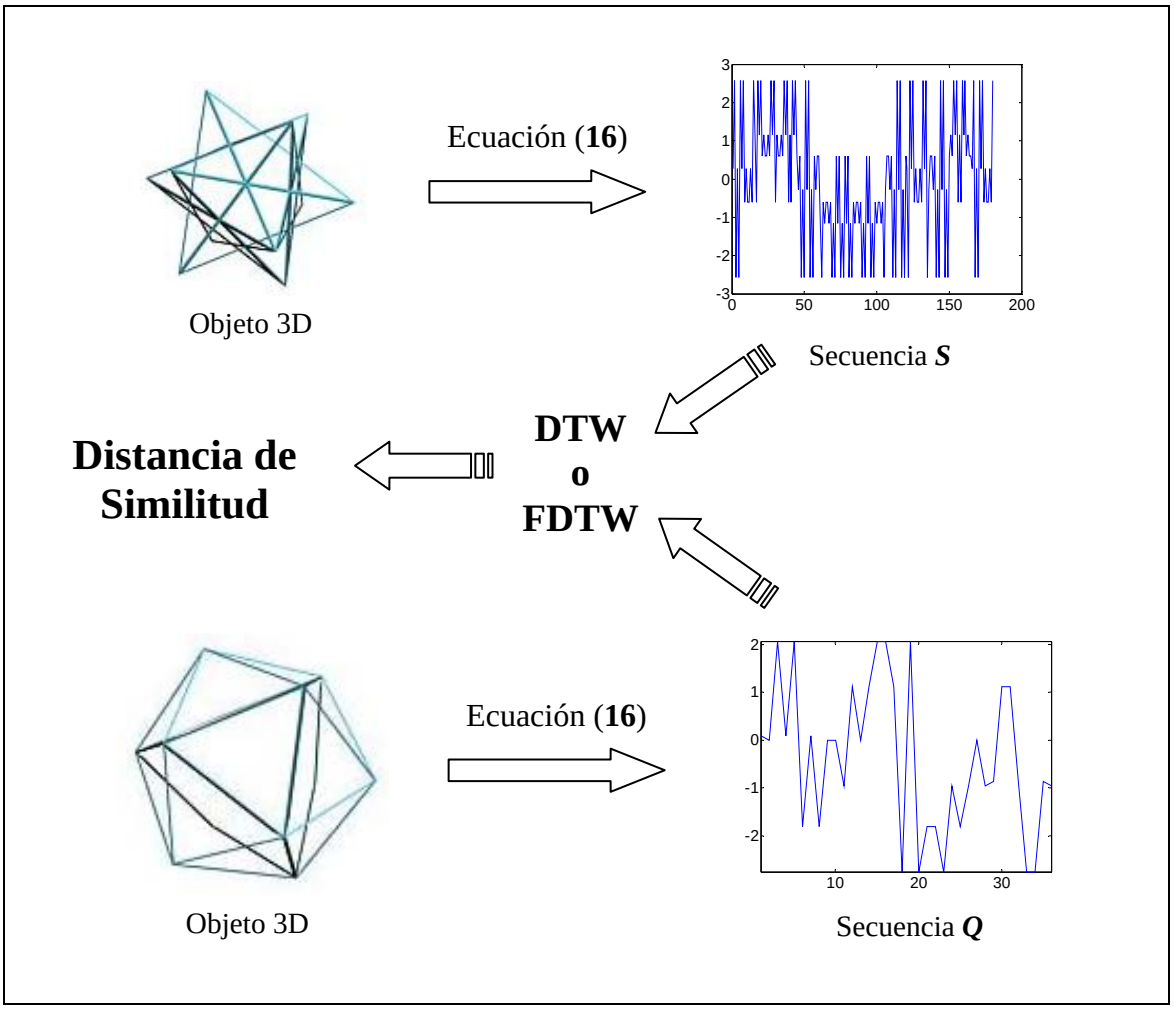


Figura 41. Cómputo de Similitud entre figuras geométricas usando secuencias representativas. Nótese que el esquema actual sólo define búsqueda secuencial.

A pesar de la pérdida aparente de información que implica la construcción de una sola secuencia por figura (a diferencia del método del capítulo anterior). Este método logra computar la similitud de entidades tridimensionales de tamaño completamente distinto. Ello se debe a la capacidad de *FDTW* de *alinear*, secuencias de diferente tamaño.

El reconocimiento de figuras similares diferenciados por transformaciones lineales básicas no presenta mayor dificultad como lo mostrarán resultados experimentales.

Hasta el momento, el método de comparación, sólo ha considerado la búsqueda secuencial. Sin embargo, se han empleado dos diferentes técnicas de indexado, en la siguiente sección se cubre dicho aspecto.

8.3 INDEXADO DE DTW

El indexado de secuencias se ha considerado como una aproximación a la búsqueda secuencial. Se compromete la precisión en búsqueda de rapidez.

Aunque existen múltiples técnicas de indexado, todas persiguen la misma meta, *representar una misma entidad en un espacio menor manejable*. Otra característica que une a los métodos de indexado es la utilización de un método de acceso multidimensional. No es parte de este trabajo detallar los aspectos relacionados con dichas estructuras multidimensionales.

En esta sección sólo preocupa la representación de las secuencias (asociadas a cada figura dentro de la base de datos), y a la adecuación de la función *MinDist*, que es implementada en una estructura multidimensional genérica, como lo es R-Tree.

En [Koegh02], se propone una técnica de reducción de dimensionalidad (**17**) (**18**) [*]basada en las restricciones globales de *DTW* clásico. Una vez que las secuencias a comparar han sido *reducidas*, pueden ser almacenadas como puntos *n*-dimensionales, dentro de una estructura como R-Tree [Guttman84]. Por lo tanto la función *MinDist*, debe ser modificada para manejar dicho puntos *n*-dimensionales.

$$U_i = \max\left(q_{i-r} : q_{i+r}\right) \; L_i = \min\left(q_{i-r} : q_{i+r}\right) \tag{17}$$

$$\begin{aligned} \overline{U}_i &= \max\left(U_{\frac{n}{N}(i-1)+1}, \ldots, U_{\frac{n}{N}(i)} \right) \\ \overline{L}_i &= \min\left(L_{\frac{n}{N}(i-1)+1}, \ldots, L_{\frac{n}{N}(i)} \right) \end{aligned} \tag{18}$$

$$MinDist\ (Q,R) = \sqrt{\sum_{i=1}^{N} \frac{n}{N} \begin{cases} \left(l_i - \overline{U_i}\right)^2, l_i > \overline{U_i} \\ \left(h_i - \overline{L_i}\right)^2, h_i > \overline{L_i} \\ 0, otherwise \end{cases}} \tag{19}$$

En la figura 42 se observa una figura (amortiguador), cuya secuencia representativa ha sido construida. Sin embargo la dimensionalidad de dicha secuencia (100 puntos aprox.) hace que la secuencia sea impráctica de indexar directamente.

Las envolventes superior (Upper) e inferior (Lower) son creadas con un rango *r* entero que determina en ancho de la banda, en este caso, obedecen a la restricción global de Shakoe-Chiba (**17**), posteriormente una reducción de dimensionalidad es aplicada a estas envolventes, donde *n* es el tamaño original de la secuencia, y *N* es el

[*] $Q = q_1, q_2, q_3, \ldots, q_i, \ldots, q_n$; $C = c_1, c_2, c_3, \ldots, c_j, \ldots, c_m$

número de *pedazos* que se desean crear, de ahí que esta técnica se titule *Piecewise Aggregate Approximation* (PAA). En la figura 42 se condensa este proceso.

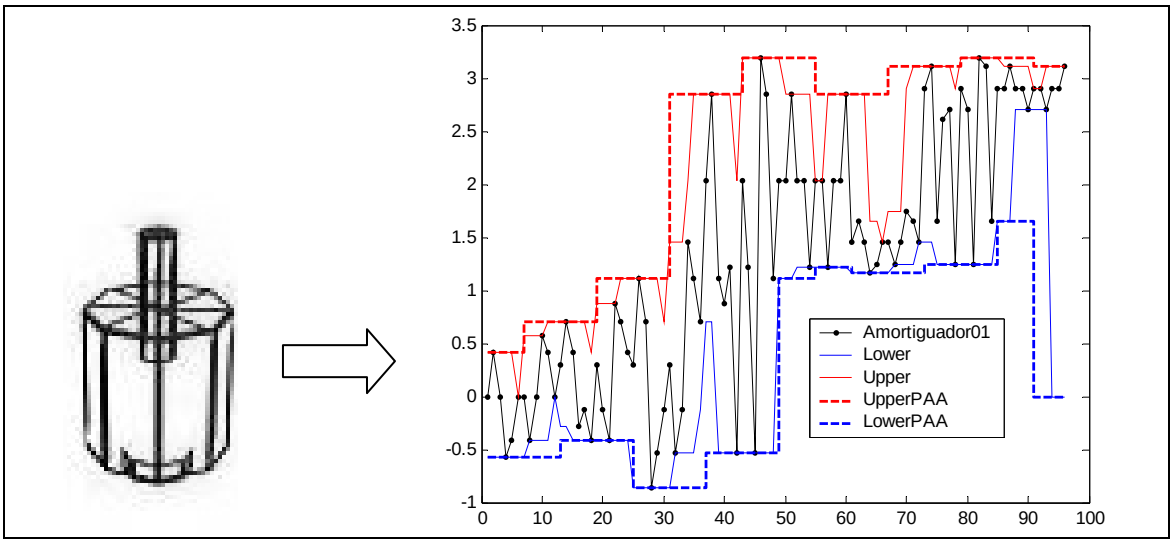


Figura 42. Reducción de dimensionalidad e indexado por PAA. Nótese que el número de pedazos (*N*) es 16.

Como se muestra en la sección de resultados, otra técnica que se utilizó para la indexación de figuras geométricas fue la propuesta en [Kim01]

Kim et al. [Kim01] propuso un método de indexado basado un vector de 4 elementos, esto es, el primer, el último, el mayor, y el menor elemento de una secuencia S. Demostró que la función LB_Kim es menor o igual que *DTW*, que obedece las propiedades de métrica, y garantiza no descartar falsamente candidatos. Detalles sobre la demostración de lo anterior se encuentran en [Kim01]. El vector de 4 elementos y LB_Kim se definen así:

$$F(S) = \langle \text{Pr}\,imero(S), Último(S), Mayor(S), Menor(S) \rangle \quad \textbf{(20)}$$

$$LB\_Kim(S,P) = \max \begin{cases} |\text{Pr}\,imero(S) - \text{Pr}\,imero(P)| \\ |Último(S) - Último(P)| \\ |Mayor(S) - Mayor(P)| \\ |Menor(S) - Menor(P)| \end{cases} \quad \textbf{(21)}$$

Aunque se han propuesto técnicas de indexado que estadísticamente superan a LB_Kim [Keogh02], en el caso de objetos 3D creados por un sistema DAC, cuyos vértices tienen un orden establecido, resulta pertinente el uso de LB_Kim para la indexación de las secuencias representativas presentadas en este trabajo, ya que permiten representar el primer y último nodo, la magnitud del nodo mayor y menor, que es, la suma de los componentes en *X*, *Y*, y *Z*. Definido (**20**) y (**21**), se implementa una estructura de datos como R-Tree, el vector *F(S)* es insertado como un punto en un espacio *k*-dimensional, en este caso, *k* es igual a 4. La función *MinDist* será igual a

LB_Kim, cuya complejidad es constante. Por último, se utiliza el algoritmo de *k* vecinos cercanos para obtener los *k* candidatos más similares, y así calcular con *DTW* la verdadera distancia final. La siguiente figura esquematiza el método para computar similitud entre figuras geométricas con *DTW*, y el módulo para indexar dichas figuras geométricas usando LB_Kim.

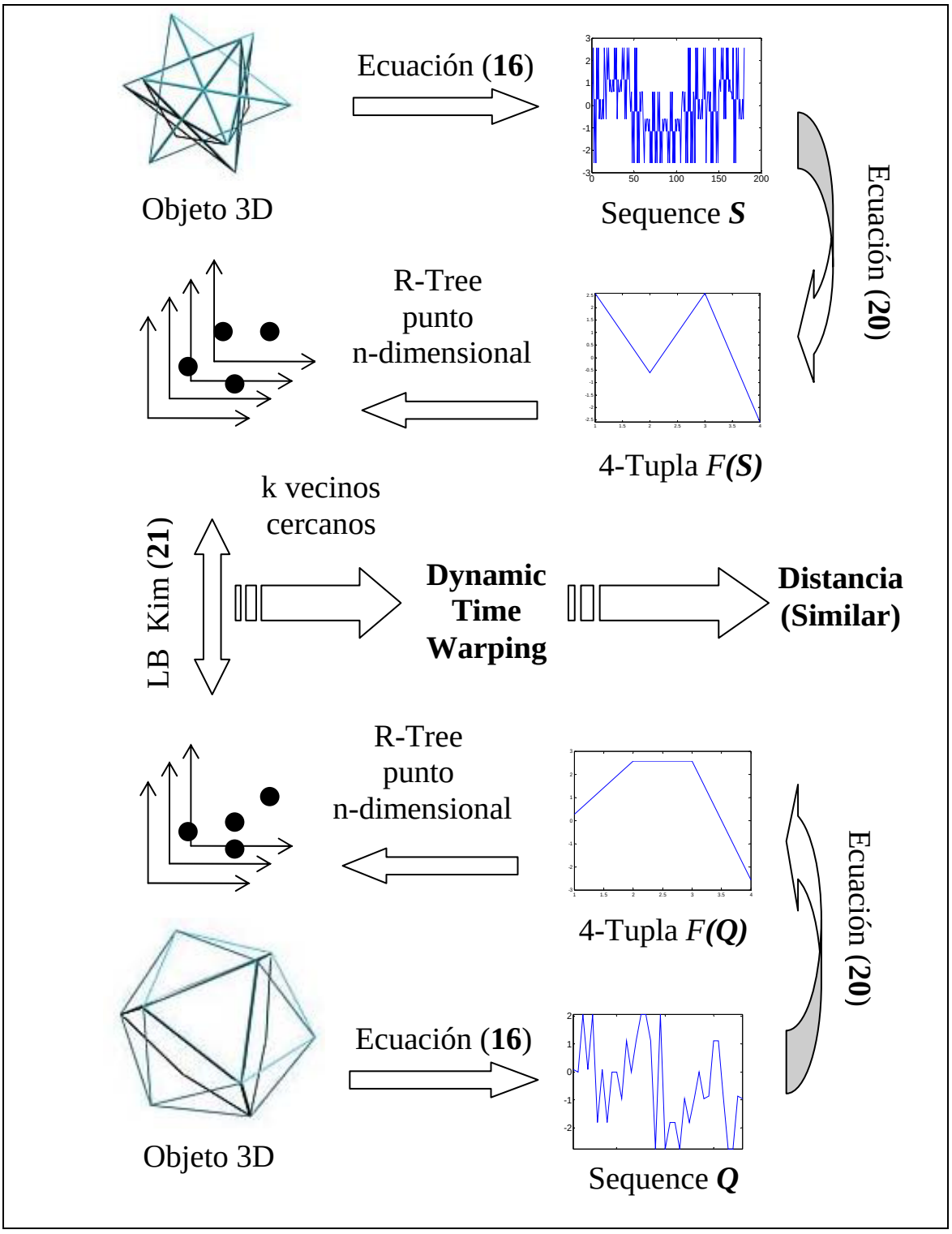


Figura 45. Representación y Cómputo de Similitud entre Poliedros. Nótese que se ilustra el caso de usar la técnica de LB_Kim.

El modelo consta de una etapa de pre-procesamiento, ésta involucra la creación de la secuencia representativa (**16**) y la reducción de dimensionalidad (**20**).

De esta forma, por cada objeto 3D en la base de datos se obtendrá una 4-tupla que se inserta como un punto 4-dimensional en la estructura de datos multidimensional, este proceso se realiza una sola vez.

Los objetos 3D almacenados en la estructura R-Tree pueden ser accedidos mediante su identificador.

Las funciones LB_Kim (**21**) y k vecinos cercanas proporcionan los *k* candidatos más similares con respecto a un objeto consulta.

Finalmente, DTW será computado con los *k* candidatos para obtener una distancia de similaridad final.

REFERENCIAS DEL CAPÍTULO

[Angeles05a] A. Angeles-Yreta, J. Figueroa-Nazuno. "Semejanza entre Objetos Tridimensionales aplicando Redes Neuronales Fuzzy ART". *Advances in Computer Science in Mexico, Research on Computer Science*. Vol. 13, A. Gelbukh y H. Calvo (Eds.), ISSN: 1665-9899, 2005, pp. 127-134.

[Angeles05b] A. Angeles-Yreta, J. Figueroa-Nazuno. "Similarity Search in 3D Objects and Dynamic Time Warping". *Proceedings of International Seminar on Computational Intelligence*. IEEE CIS Mexico Chapter. 2005.

[Angeles05c] A. Angeles-Yreta, J. Figueroa-Nazuno. "Búsqueda de Semejanza entre Objetos Tridimensionales por Indexado". *Congreso Nacional de Física 2005*. ISSN 0187-4713. 2005. pp. 97

[Angeles05d] A. Angeles-Yreta, J. Figueroa-Nazuno. "Computing Similarity Among 3D Objects Using Dynamic Time Warping". *Lecture Notes in Computer Science* (Springer-Verlag), Progress in Pattern Recognition, Image Analysis and Applications: *10th Iberoamerican Congress on Pattern Recognition, CIARP 2005*, Havana, Cuba, November 15-18, 2005, ISSN: 0302-9743, ISSN: 0302-9743. pp. 319-326.

[Angeles05e] A. Angeles-Yreta, J. Figueroa-Nazuno, K. Ramírez-Amaro. "Búsqueda de Semejanza entre Objetos 3D por Indexado". *Decimosexta Reunión de Otoño Comunicaciones, Computación* ROC&C 2005. 2005.

[Deller00] Deller J. R., Hansen J. H. L, Proakis J. G. *Discrete-Time Processing of Speech Signals*. Wiley-IEE Press, ISBN: 0780353862, Second Edition, pp 623-676, 2000.

[Kim01] Kim S. W., Park S., Chu W. W. "An Index-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Database". *In Proceedings of the 17th International Conference on Data Engineering (ICDE'01)*, pp. 607-614, 2001.

[Jang00] Jang J.S.R, Gao M.Y. "A Query-by-Singing System based on Dynamic Programming". *International Workshop on Systems Resolutions* (the 8th Bellman Continuum), pp. 85-89, 2000.

[Koegh02] Keogh E. "Exact Indexing of Dynamic Time Warping". *In Proceedings of the 28th VLDB Conference*, pp. 406-417, 2002.

[Guttman84] Guttman A. “R-Trees: A Dynamic Index Structure for Spatial Searching”. *In Proceedings of ACM SIGMOD’84 Conference on Management of Data*, ACM Press, pp. 47-57, 1984.

9 RESULTADOS

Es objetivo de este capítulo detallar los resultados experimentales del presente trabajo. Las pruebas experimentales se categorizan en dos ramas:

1. Resultados de pruebas experimentales del método para representar y computar similitud entre figuras geométricas usando *RNA's Fuzzy ART* (capítulo 7).
2. Resultados de pruebas experimentales del método para representar y computar similitud entre figuras geométricas usando *Dynamic Time Warping* y *Fast Dynamic Time Warping* (capítulo 8).

El conjunto de datos (base de figuras) se detalla en capítulo 6, dicho conjunto se compone de: a) figuras generadas a partir de plantillas de un sistema DAC típico, y b) figuras generadas de forma suedo-aleatoria a partir de figuras base.

9.1 REPRESENTACIÓN Y CÓMPUTO DE SIMILITUD USANDO RNA FUZZY ART

En esta sección se muestran resultados e interpretaciones publicadas en [Angeles05a]. Para este experimento fueron considerados 46 figuras, creados a partir de 13 platillas predefinidas de 3D Studio MAX (chafCil, ext-c, ext-l, huso, genon, poliedro, prisma, amortiguador, pirámide, muelle, cono, toriode, tubo). El tamaño (longitud del índice de vértices) de las figuras es variable, como las *RNA Fuzzy ART* consideran arreglos de longitud constante, se utilizó la función de MATLAB `resample`, para redimensionar la longitud de los índices de vértices. Los parámetros para las 3 RNA Fuzzy ART empleadas (ver capítulo 7), fueron:

ρ (rho)	α (alpha)	β (beta)
[0.9, 0.7]	**0.1**	**1**

Siendo alpha y beta parámetros fijos, dichos parámetros son los comúnmente usados en la literatura. Se procedió a generar resultados variando el parámetro de vigilancia en decrementos de 0.05, a partir de 0.9. Establecer el parámetro de vigilancia a 1 indicaría que el clasificador asigna a cada patrón de entrada distinto una categoría distinta, es

decir, proliferación de clases. De forma similar, establecer el parámetro de vigilancia menor a 0.7 puede generar clases demasiado *permisivas*.

Tabla 2. Clasificación de figuras geométricas, ρ = 0.85

	ARTx		ARTy		ARTz	Clase
ChafCil01	0	ChafCil01	0	ChafCil01	0	**0**
ChafCil02	0	ChafCil02	1	ChafCil02	0	**0**
ChafCil03	0	ChafCil03	1	ChafCil03	0	**0**
ChafCil04	0	ChafCil04	1	ChafCil04	0	**0**
ChafCil05	1	ChafCil05	2	ChafCil05	1	**1**
ChafCil06	2	ChafCil06	2	ChafCil06	2	**2**
Ext_C01	3	Ext_C01	3	Ext_C01	3	**3**
Ext_C02	3	Ext_C02	3	Ext_C02	3	**3**
Ext_C03	3	Ext_C03	3	Ext_C03	3	**3**
Ext_C04	4	Ext_C04	3	Ext_C04	3	**4**
Ext_C05	4	Ext_C05	3	Ext_C05	3	**4**
Ext_C06	4	Ext_C06	3	Ext_C06	3	**4**
Ext_L01	5	Ext_L01	4	Ext_L01	4	**5**
Ext_L02	5	Ext_L02	4	Ext_L02	4	**5**
Ext_L03	5	Ext_L03	4	Ext_L03	4	**5**
Ext_L04	6	Ext_L04	4	Ext_L04	4	**5**
Ext_L05	5	Ext_L05	4	Ext_L05	4	**5**
Gengon01	7	Gengon01	5	Gengon01	5	**6**
Gengon02	8	Gengon02	6	Gengon02	5	**7**
Gengon03	7	Gengon03	3	Gengon03	6	**8**
Gengon04	8	Gengon04	7	Gengon04	6	**9**
Gengon05	9	Gengon05	6	Gengon05	7	**10**
Huso01	10	Huso01	0	Huso01	8	**11**
Huso02	11	Huso02	8	Huso02	9	**12**
Huso03	4	Huso03	3	Huso03	3	**4**
Huso04	4	Huso04	3	Huso04	3	**4**
Poliedro19	12	Poliedro19	9	Poliedro19	10	**13**
Poliedro20	13	Poliedro20	10	Poliedro20	11	**14**
Prisma01	14	Prisma01	11	Prisma01	12	**15**
Prisma02	14	Prisma02	11	Prisma02	12	**15**
Prisma03	15	Prisma03	11	Prisma03	12	**16**
Prisma04	15	Prisma04	11	Prisma04	12	**16**
Prisma05	16	Prisma05	11	Prisma05	12	**17**
Amortiguador01	17	Amortiguador01	12	Amortiguador01	13	**18**
Amortiguador02	18	Amortiguador02	12	Amortiguador02	14	**19**
Cono01	1	Cono01	13	Cono01	1	**20**
Cono02	1	Cono02	13	Cono02	1	**21**
Muelle01	19	Muelle01	13	Muelle01	15	**22**
Piramide01	20	Piramide01	14	Piramide01	16	**23**
Piramide02	20	Piramide02	14	Piramide02	16	**23**
Toroide01	21	Toroide01	15	Toroide01	17	**24**
Toroide02	21	Toroide02	15	Toroide02	17	**24**
Tubo01	21	Tubo01	16	Tubo01	17	**25**
Tubo02	21	Tubo02	16	Tubo02	17	**25**

Tabla 3. Clasificación de figuras geométricas, ρ = 0.8

	ARTX		ARTY		ARTZ	CLASE
ChafCil01	0	ChafCil01	0	ChafCil01	0	**0**
ChafCil02	0	ChafCil02	0	ChafCil02	0	**0**
ChafCil03	0	ChafCil03	0	ChafCil03	0	**0**
ChafCil04	0	ChafCil04	1	ChafCil04	0	**0**
ChafCil05	1	ChafCil05	1	ChafCil05	1	**1**
ChafCil06	2	ChafCil06	1	ChafCil06	2	**2**
Ext_C01	3	Ext_C01	2	Ext_C01	3	**3**
Ext_C02	3	Ext_C02	2	Ext_C02	3	**3**
Ext_C03	3	Ext_C03	2	Ext_C03	3	**3**
Ext_C04	3	Ext_C04	2	Ext_C04	3	**3**
Ext_C05	3	Ext_C05	2	Ext_C05	3	**3**
Ext_C06	3	Ext_C06	2	Ext_C06	3	**3**
Ext_L01	4	Ext_L01	3	Ext_L01	3	**4**
Ext_L02	4	Ext_L02	3	Ext_L02	4	**5**
Ext_L03	4	Ext_L03	3	Ext_L03	4	**5**
Ext_L04	4	Ext_L04	3	Ext_L04	4	**5**
Ext_L05	4	Ext_L05	3	Ext_L05	4	**5**
Gengon01	5	Gengon01	3	Gengon01	5	**6**
Gengon02	5	Gengon02	4	Gengon02	5	**7**
Gengon03	6	Gengon03	2	Gengon03	5	**8**
Gengon04	5	Gengon04	4	Gengon04	6	**9**
Gengon05	6	Gengon05	5	Gengon05	6	**9**
Huso01	7	Huso01	5	Huso01	7	**10**
Huso02	8	Huso02	6	Huso02	8	**11**
Huso03	3	Huso03	2	Huso03	3	**3**
Huso04	3	Huso04	2	Huso04	3	**3**
Poliedro19	9	Poliedro19	7	Poliedro19	9	**12**
Poliedro20	10	Poliedro20	8	Poliedro20	10	**13**
Prisma01	11	Prisma01	9	Prisma01	11	**14**
Prisma02	11	Prisma02	9	Prisma02	11	**14**
Prisma03	12	Prisma03	9	Prisma03	11	**15**
Prisma04	12	Prisma04	9	Prisma04	11	**15**
Prisma05	12	Prisma05	9	Prisma05	11	**15**
Amortiguador01	7	Amortiguador01	10	Amortiguador01	7	**16**
Amortiguador02	13	Amortiguador02	10	Amortiguador02	12	**17**
Cono01	1	Cono01	11	Cono01	1	**18**
Cono02	1	Cono02	11	Cono02	1	**18**
Muelle01	14	Muelle01	11	Muelle01	13	**19**
Piramide01	15	Piramide01	12	Piramide01	14	**20**
Piramide02	15	Piramide02	12	Piramide02	14	**20**
Toroide01	16	Toroide01	13	Toroide01	15	**21**
Toroide02	16	Toroide02	13	Toroide02	15	**21**
Tubo01	16	Tubo01	14	Tubo01	15	**22**
Tubo02	16	Tubo02	14	Tubo02	15	**22**

La tripleta de *RNA's Fuzzy ART*, permiten una clasificación *significativa* de patrones de entrada con un valor estándar de 0.8, valor recomendado en la literatura.

Tabla 4. Clasificación de figuras geométricas, ρ = 0.7

	ARTx		ARTy		ARTz	Clase
ChafCil01	0	ChafCil01	0	ChafCil01	0	**0**
ChafCil02	0	ChafCil02	0	ChafCil02	0	**0**
ChafCil03	0	ChafCil03	0	ChafCil03	0	**0**
ChafCil04	0	ChafCil04	0	ChafCil04	0	**0**
ChafCil05	1	ChafCil05	0	ChafCil05	1	**1**
ChafCil06	2	ChafCil06	1	ChafCil06	2	**2**
Ext_C01	3	Ext_C01	1	Ext_C01	3	**3**
Ext_C02	3	Ext_C02	1	Ext_C02	3	**3**
Ext_C03	3	Ext_C03	1	Ext_C03	3	**3**
Ext_C04	3	Ext_C04	1	Ext_C04	3	**3**
Ext_C05	3	Ext_C05	1	Ext_C05	3	**3**
Ext_C06	3	Ext_C06	1	Ext_C06	3	**3**
Ext_L01	4	Ext_L01	2	Ext_L01	3	**4**
Ext_L02	4	Ext_L02	2	Ext_L02	3	**4**
Ext_L03	4	Ext_L03	2	Ext_L03	3	**4**
Ext_L04	4	Ext_L04	2	Ext_L04	3	**4**
Ext_L05	4	Ext_L05	2	Ext_L05	3	**4**
Gengon01	1	Gengon01	2	Gengon01	1	**1**
Gengon02	5	Gengon02	2	Gengon02	2	**5**
Gengon03	5	Gengon03	1	Gengon03	4	**6**
Gengon04	5	Gengon04	3	Gengon04	4	**6**
Gengon05	5	Gengon05	3	Gengon05	4	**6**
Huso01	2	Huso01	3	Huso01	5	**7**
Huso02	6	Huso02	4	Huso02	5	**8**
Huso03	3	Huso03	1	Huso03	3	**3**
Huso04	3	Huso04	1	Huso04	3	**3**
Poliedro19	6	Poliedro19	5	Poliedro19	6	**9**
Poliedro20	7	Poliedro20	5	Poliedro20	6	**9**
Prisma01	7	Prisma01	4	Prisma01	7	**10**
Prisma02	8	Prisma02	4	Prisma02	7	**11**
Prisma03	8	Prisma03	4	Prisma03	7	**11**
Prisma04	8	Prisma04	4	Prisma04	7	**11**
Prisma05	8	Prisma05	4	Prisma05	7	**11**
Amortiguador01	9	Amortiguador01	6	Amortiguador01	7	**12**
Amortiguador02	9	Amortiguador02	6	Amortiguador02	8	**13**
Cono01	9	Cono01	6	Cono01	8	**13**
Cono02	10	Cono02	6	Cono02	8	**14**
Muelle01	11	Muelle01	6	Muelle01	9	**15**
Piramide01	10	Piramide01	7	Piramide01	10	**16**
Piramide02	10	Piramide02	7	Piramide02	10	**16**
Toroide01	12	Toroide01	8	Toroide01	10	**17**
Toroide02	12	Toroide02	8	Toroide02	10	**17**
Tubo01	12	Tubo01	9	Tubo01	11	**18**
Tubo02	12	Tubo02	9	Tubo02	11	**18**

En la figura 46 y 47, se muestran 19 clases, contiendo un total de 46 figuras. Por ejemplo, en la clase 0, se agrupan *chafcils*, con un número de segmentos relativamente bajo, la clase 1, agrupa dos objetos con estructuras similares, la clase 3 comprende de todos los objetos ext-c, y de un par de figuras (husos), que son incluidos en esta clase

debido a que se redimensionaron los índices de vértices de las 46 figuras clasificar. Esta misma razón influye en la clase 12, donde un amortiguador y un cono son pertenecen a la misma clase.

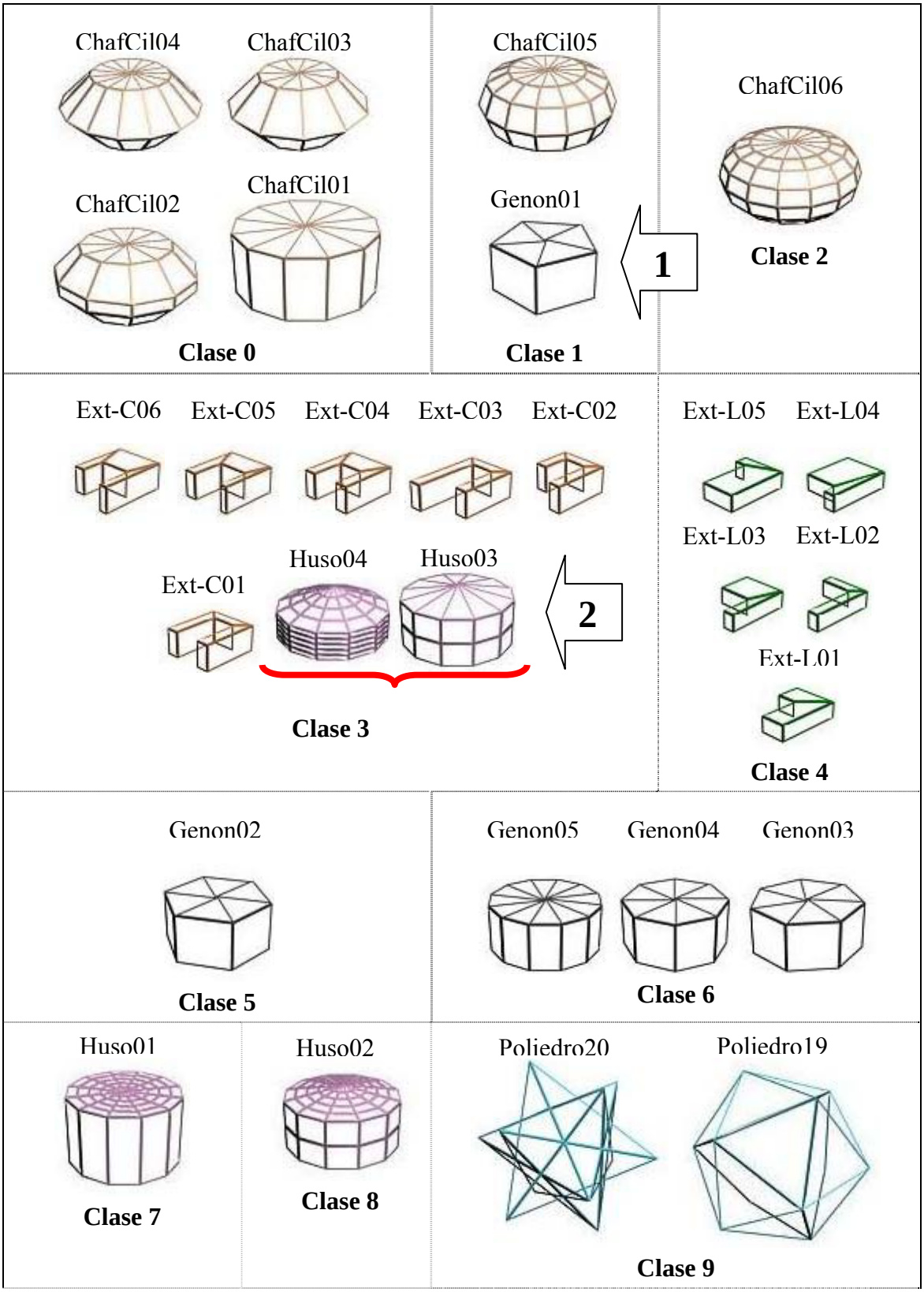


Figura 46. Clasificación de figuras geométricas DAC con RNA Fuzzy ART_1.

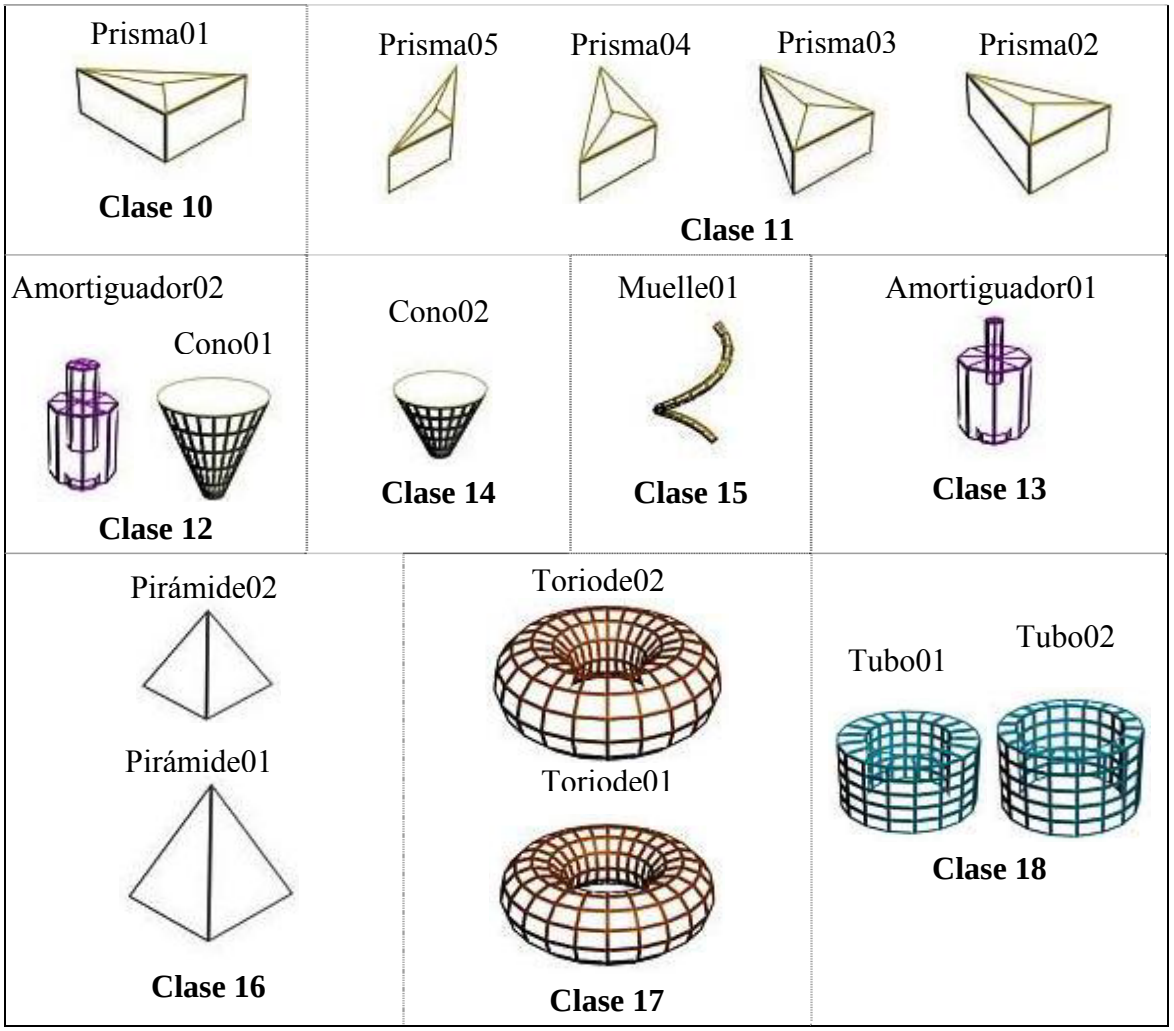


Figura 47. Clasificación de figuras geométricas DAC con RNA Fuzzy ART_2.

Este primer experimento esquematizó la idea de representar y determinar semejanza entre figuras de tamaño similar. El siguiente experimento consistió en la clasificación de 24 figuras seudo-aleatorias, las figuras en cuestión son de tamaño similar. Las figuras fueron generadas de forma seudo-aleatoria a partir de figuras base. Los resultados muestran la capacidad del método para computar semejanza considerando la forma geométrica, y pequeñas variaciones. Los valores de parámetros que se emplearon fueron $\rho = 0.9$, $\alpha = 0.1$, y $\beta = 1$.

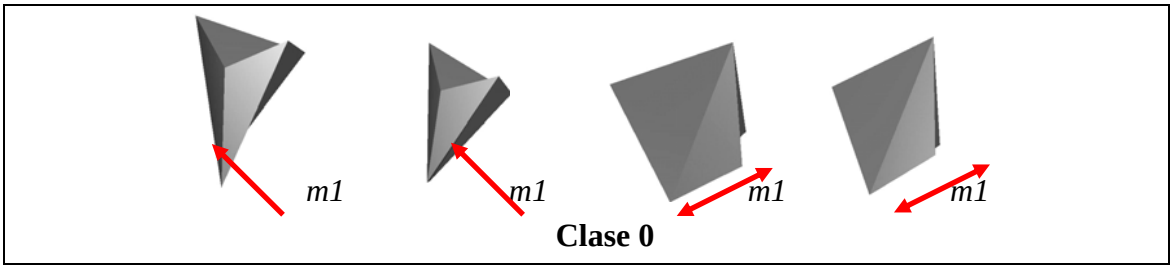


Figura 48. Clasificación de figuras aleatorias con RNA Fuzzy ART_1.

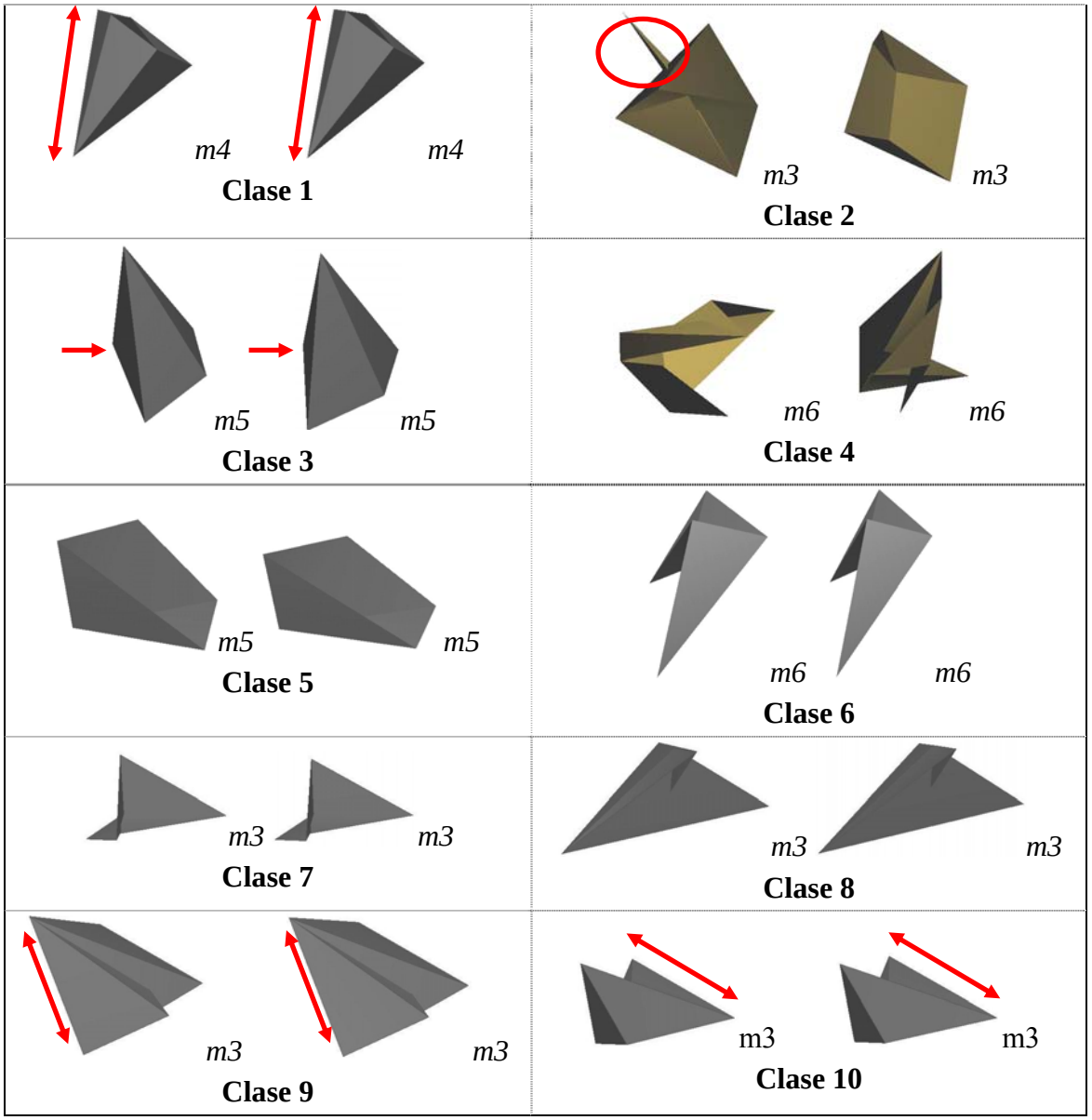


Figura 49. Clasificación de figuras aleatorias con RNA Fuzzy ART_2.

Las flechas (sobre las figuras), son de longitud constante, con el objetivo de denotar diferentes tamaños de figuras. En algunas ocasiones los círculos denotan presencias y ausencias entre objetos. Adyacente a la figura se encuentra el número de modificaciones realizadas a la figura base para crear el objeto en cuestión (ver capítulo 6, creación de figuras seudo-aleatorios), por ejemplo *m4* denota 4 vértices **han sido modificados**.

A continuación se muestra otro experimento realizado con los mismos valores de parámetro y otro conjunto de figuras seudo-aleatorias, esto es $\rho = 0.9$, $\alpha = 0.1$, y $\beta = 1$.

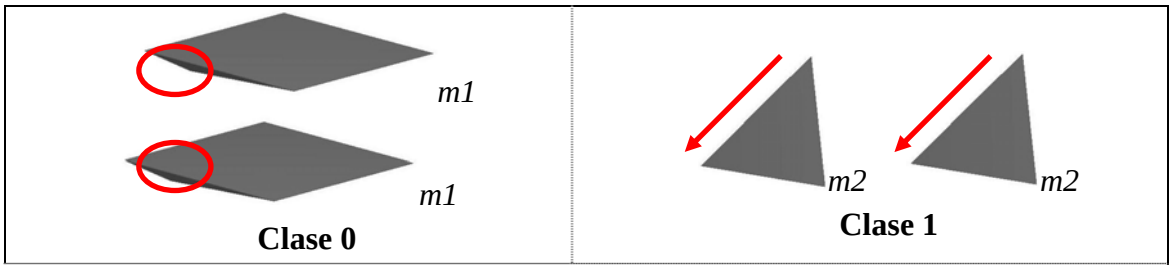
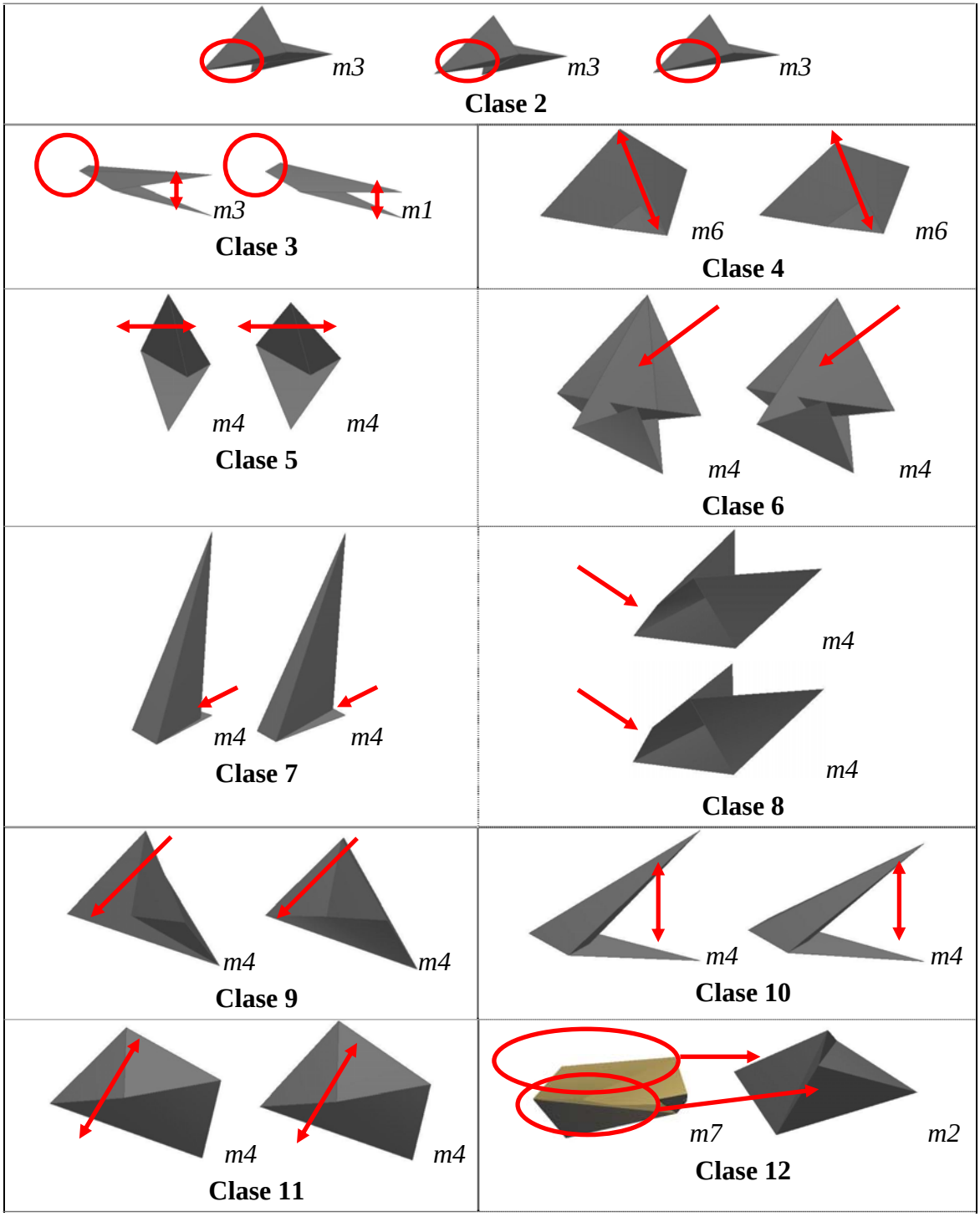


Figura 50. Clasificación de figuras aleatorias con RNA Fuzzy ART_3.



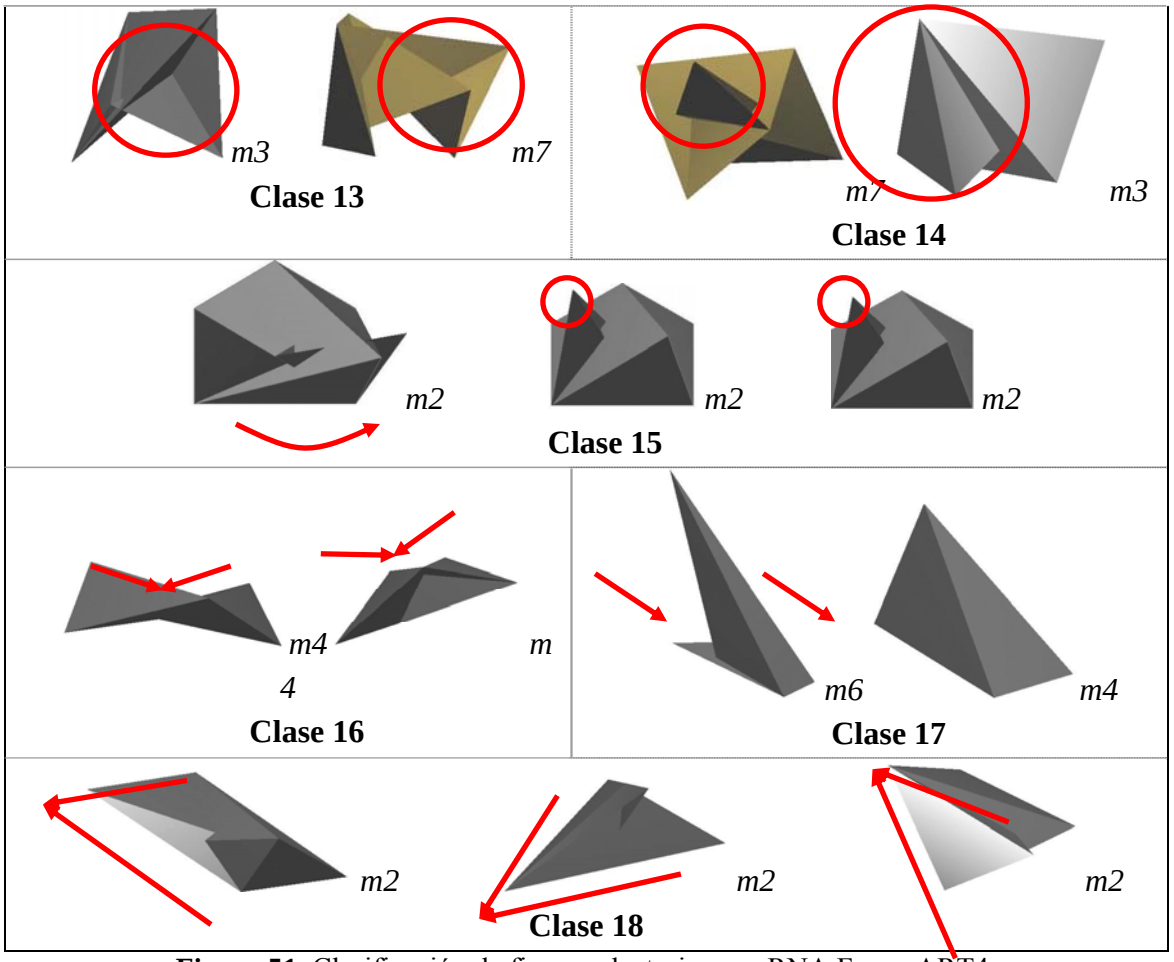


Figura 51. Clasificación de figuras aleatorias con RNA Fuzzy ART4.

Como el lector ya podrá detectar, no todos las figuras seudo-aleatorias son figuras cerradas o sólidas.

Pequeñas y grandes variaciones pueden ser detectadas por el método, la potencia de la *RNA Fuzzy ART*, es aprovechada.

El parámetro de vigilancia es determinante para la creación de más o menos clases, la verificación de las clases la hace el usuario experto, ya que no hay un criterio universal para determinar qué valor de *rho* es el óptimo.

9.2 REPRESENTACIÓN Y CÓMPUTO DE SIMILITUD USANDO DTW

En esta sección se presentan algunos resultados obtenidos de pruebas experimentales con un conjunto de datos similar a los empleados en la sección anterior.

Anteriormente se ha mencionado que el método de similitud basado en *DTW* es una evolución del método basado en *RNA Fuzzy ART* (ambos propuestos en este trabajo).

Existen varias razones que sugieren una mayor aplicabilidad, por ejemplo, la verificación dinámica de firmas.

Sin embargo, son los resultados de las pruebas experimentales que dan cuenta de las ventajas de un método sobre el otro. La tabla que a continuación se lista muestra una matriz de distancia de todas las figuras de una base de datos contra sí mismos. Este tipo de experimentos exhaustivos permiten observar la eficacia de la métrica de similitud. Dicho conjunto de figuras consiste de más de 60 elementos, por practicidad se ha recortado, resultados completos se encuentran en el anexo.

Tabla 5.Distancias DTW entre figuras geométricas[1]. *Poliedros DAC*.

Gengon01		***Huso01***		***Poliedro09***		***Piramide01***	
Gengon01	0	Huso01	0	Poliedro09	0	Piramide01	0
Gengon02	13.7403	Huso02	130.4449	Poliedro05	96.3107	Piramide02	8.684
Gengon03	20.4739	ChafCil05	205.8659	Poliedro02	113.5916	Poliedro03	21.508
Gengon04	26.0433	ChafCil02	221.6002	Poliedro07	114.3981	Poliedro07	24.506
Prisma04	29.5395	ChafCil03	223.1862	Gengon04	118.0033	Poliedro04	24.762
Prisma05	31.4265	Huso04	230.8463	Poliedro12	118.7566	Prisma04	33.084
Ext_L04	35.497	ChafCil04	234.2145	Gengon02	121.3382	Prisma03	34.2812
Ext_L01	35.7141	ChafCil01	234.6597	Gengon01	121.6394	Prisma02	34.8248
Piramide02	36.3354	ChafCil06	236.8923	Gengon03	124.1357	Prisma01	37.1623
Prisma03	36.6871	Gengon05	239.8136	Prisma02	126.329	Poliedro08	38.12
Ext_L03	37.4308	Huso03	248.5881	Piramide02	127.7083	Prisma05	40.9303
Ext_C01	37.9528	Ext_C06	263.1274	Prisma03	129.9838	Ext_L04	44.3704
Ext_L05	37.9665	Ext_C01	267.2925	Ext_C03	133.2251	Ext_L01	48.7202
Ext_C04	38.4054	Ext_C02	268.3045	Gengon05	133.3867	Ext_L03	48.8907
Ext_C05	39.2078	Ext_C03	274.9795	Poliedro11	134.6425	Gengon01	49.0548
Ext_C02	39.8385	Ext_C05	276.5467	Prisma01	136.4083	Ext_L05	51.0327
Ext_L02	40.4172	Ext_C04	279.9909	Piramide01	144.0109	Ext_L02	54.0821
Ext_C03	40.6323	Ext_L02	284.0094	Poliedro06	145.9328	Poliedro11	57.1539
Poliedro07	40.8918	Gengon03	292.6591	Ext_C04	148.1046	Gengon02	58.5536
Ext_C06	43.2608	Gengon04	292.757	Ext_C05	149.2436	Ext_C04	59.3464

La tabla anterior muestra la distancia calculada con *DTW*, dicha distancia es calculada con el método propuesto en el capítulo 8. En la figura 52, se ilustra la relación de distancia establecida entre figuras geométricas correspondiente a las tablas 2 y 3.

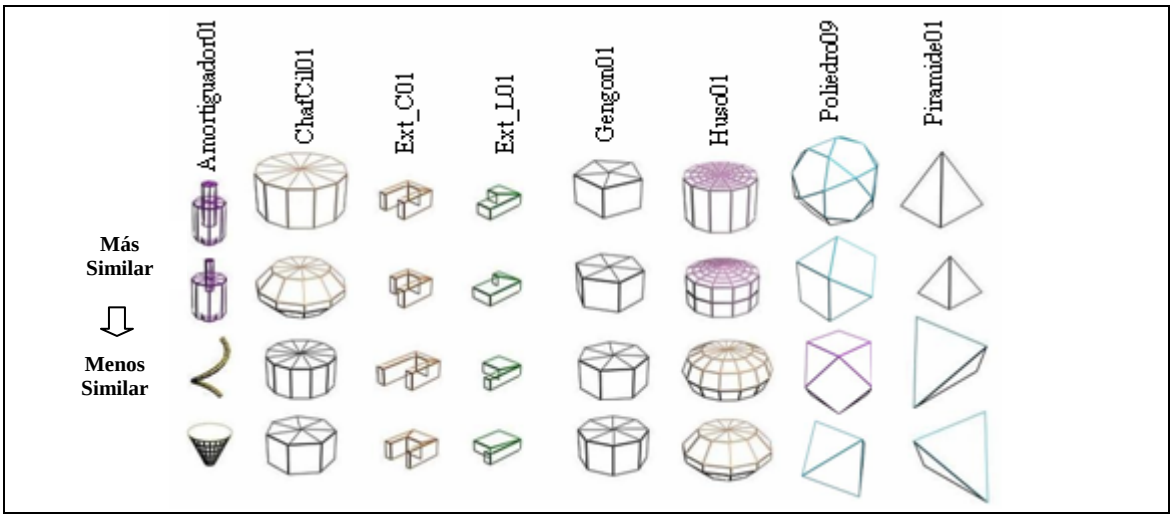


Figura 52.Distancias DTW entre figuras geométricas[1]. *Poliedros DAC*. Figuras correspondientes a las tablas 5 y 6,

Tabla 6.Distancias DTW entre figuras geométricas[2]. *Poliedros DAC*.

Amortiguador02		***ChafCil01***		***Ext_C01***		***Ext_L01***	
Amortiguador02	0	ChafCil01	0	Ext_C01	0	Ext_L01	0
Amortiguador01	118.861	ChafCil02	106.2858	Ext_C02	6.672	Ext_L05	5.0388
Muelle01	227.2536	Gengon05	120.635	Ext_C03	7.5414	Ext_L03	5.7651
Cono02	235.3789	Gengon03	123.7277	Ext_C05	9.3554	Ext_L04	11.6447
Ext_L02	237.133	ChafCil03	126.5888	Ext_C04	9.7154	Ext_L02	12.33
Gengon04	237.3816	Gengon04	127.5282	Ext_C06	16.767	Prisma04	19.6203
Prisma05	238.0577	Gengon02	130.556	Ext_L03	23.4665	Ext_C02	21.283
Gengon01	242.2867	Huso03	134.7352	Ext_L02	23.9167	Ext_C04	23.8427
Gengon03	245.7087	Gengon01	138.4957	Ext_L04	24.6086	Ext_C06	24.3526
Ext_L04	246.7774	ChafCil04	145.035	Prisma04	27.7292	Ext_C05	26.2439
Ext_L03	247.6925	Ext_C03	165.9906	Ext_L01	28.0226	Ext_C01	28.0226
Ext_L01	247.8301	Poliedro02	167.751	Ext_L05	30.0278	Prisma05	29.3583
Gengon02	250.2158	Ext_C01	169.9815	Prisma05	34.5495	Ext_C03	34.9766
Gengon05	252.3078	Poliedro09	170.1852	Gengon02	37.7861	Gengon01	35.7141
Prisma04	253.4013	Ext_C06	173.6845	Gengon01	37.9528	Piramide02	36.9747
Ext_L05	255.5459	Ext_C05	174.5533	Gengon03	40.1822	Gengon02	39.3798
Poliedro03	262.9092	Ext_L04	174.932	Gengon04	44.6713	Prisma03	40.4178
Ext_C06	263.2837	Ext_C04	175.8565	Prisma03	44.7066	Poliedro03	41.6435
Prisma03	268.897	Ext_C02	179.3359	Piramide02	52.3138	Poliedro04	42.2109
Ext_C02	269.1696	Prisma04	180.7534	Prisma02	55.3076	Gengon03	43.4375

Algunos de éstos resultados se pueden localizar en [Angeles05b] [Angeles05c] [Angeles05d] [Angeles05e].

Cómo el lector podrá observar, la creación de las distancias se realiza por medio de una búsqueda secuencial. En [Angeles05b] se presentó la implementación de la técnica LB_Keogh, para la reducción de dimensionalidad e indexado de las secuencias

representativas. La siguiente tabla muestra los resultados obtenidos con 5 vecinos cercanos para la búsqueda de similitud indexada, tal y como se muestra en la sección 8.3 del capítulo anterior. Las secuencias son reducidas a 16 *pedazos*, es decir, $N = 16$. Ver ecuaciones (**17**), (**18**) y (**19**) del capítulo anterior.

Tabla 7. 5 vecinos cercanos, usando LB_Keogh sobre figuras *DAC*.

Consulta	Amortiguador01	Consulta	ChafCil01	Consulta	Ext_C01	Consulta	Ext_L01
Amortiguador01	0	ChafCil01	0	Ext_C01	0	Ext_L01	0
Amortiguador02	0.212	ChafCil02	0.5	Ext_C04	0.03031	Ext_L05	0.23
Cono02	0.5074	ChafCil04	0.6252	Ext_C05	0.12	Ext_L03	0.33
Muelle01	0.98356	ChafCil05	0.6252	Ext_C06	0.251	Ext_L04	0.33
Cono01	1.0586	ChafCil03	0.6252	Ext_C03	0.4365	Ext_L02	0.33

Consulta	Gengon01	Consulta	Huso01	Consulta	Poliedro09	Consulta	Piramide01
Gengon01	0	Huso01	0	Poliedro09	0	Piramide01	0
Gengon05	0.1155	Huso02	0.2	Poliedro10	0.275	Poliedro07	0.262
Gengon02	0.1155	Huso03	0.2	Poliedro12	0.5453	Poliedro19	0.5083
Gengon03	0.1584	Huso04	0.41	Poliedro06	0.651	Poliedro15	0.7083
Gengon04	0.1681	Prisma04	0.7359	Poliedro11	0.7047	Piramide02	0.811

Como se ha mencionado, el objetivo de cualquier técnica de indexado persigue ser una aproximación a la búsqueda secuencial, es decir, se desea que el orden que establece la distancia entre dos figuras, sea idéntico al orden que se obtiene en la búsqueda secuencial.

El siguiente conjunto de tablas y figuras ilustran lo anterior.

Tabla 8. Distancias DTW entre figuras geométricas.

	Tubo02		***Toroide01***		***Gengon05***		***Ext_C01***
Tubo02	0	Toroide01	0	Gengon05	0	Ext_C01	0
Tubo01	152.1531	Toroide02	250.1861	Gengon04	29.3092	Ext_C02	6.672
Huso03	552.5048	Tubo02	593.3119	Gengon02	35.8998	Ext_C03	7.5414
Gengon05	566.3219	Tubo01	598.7917	Gengon03	36.6878	Ext_C05	9.3554

La tabla 6, muestra distancias *DTW* de las figuras geométricas ilustradas en la figura 53, sólo se presenta el nombre del primer elemento por simplicidad, el primer elemento resulta ser la referencia y el resto las muestras.

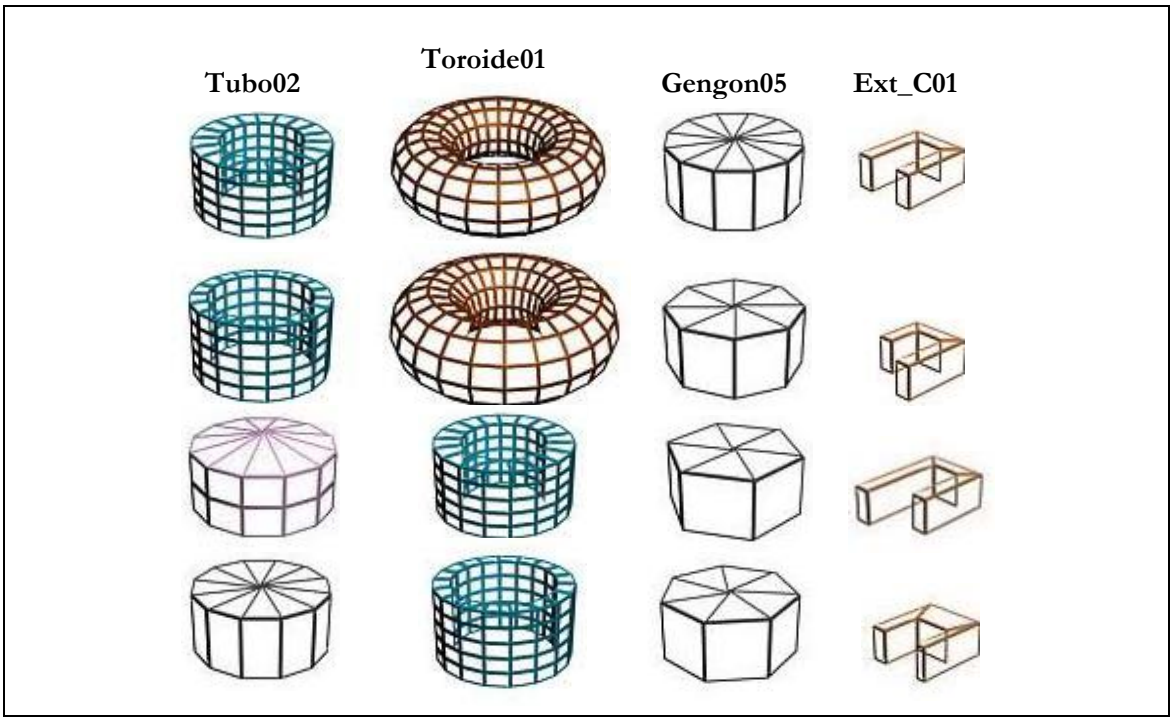


Tabla 53. Distancias DTW entre figuras geométricas$_2$.

Posteriormente se realizó la reducción de dimensionalidad y se almacenó en una estructura R-Tree, los resultados de la tabla X, muestran los 5 vecinos más cercanos con respecto a una *consulta*, dicha consulta esta denotada por el primer elemento de la columna, en la tabla 10, se ha sombreado el orden de las figuras en las que coincide la búsqueda secuencial y la búsqueda indexada.

Tabla 9. 5 vecinos cercanos, usando LB_Keogh y objetos *DAC*.

KNN to Tubo02	***KNN to Toroide01***	***KNN to Gengon05***	***KNN to Ext_C01***
Poliedro14	Toroide01	Gengon02	Ext_C01
Toroide01	Toroide02	Gengon04	Ext_C03
Toroide02	Poliedro14	Gengon05	Ext_C04
Tubo02	Poliedro10	Poliedro15	Poliedro15
Tubo01	Poliedro05	Poliedro18	Poliedro20

De forma análoga se presentan tablas y figuras de búsqueda secuencial con la función *DTW* para figuras seudo-aleatorias.

Tabla 10. Distancias DTW entre figuras seudo-aleatorias$_1$.

object0		***object12***		***object20***		***object24***
object0	0	object12	0	object20	0	object24	0
object1	6.4186	object9	13.284	object24	7.4704	object20	7.4704
object2	6.4447	object11	16.5276	object19	10.383	object19	11.1993
object4	14.0181	object13	20.1583	object18	10.4272	object18	11.2435
object5	16.3548	object8	23.4239	object17	19.8326	object23	17.9466
object7	17.1489	object46	25.489	object23	25.417	object17	20.6489
object3	17.3999	object14	26.462	object21	27.164	object21	27.7602
object6	19.0301	object45	26.611	object27	33.4492	object6	34.416
object10	24.2072	object52	26.6345	object30	36.0437	object27	35.4379
object13	25.0177	object47	26.7028	object6	36.7294	object0	36.6052
object16	28.1709	object44	27.1445	object26	37.8093	object4	37.0722
object45	30.1403	object49	27.4048	object22	37.9605	object1	37.0805
object41	33.2362	object50	28.3683	object1	38.2174	object7	37.09
object42	33.246	object51	28.4409	object2	38.2565	object2	37.1197
object44	33.2779	object10	28.4547	object0	38.7036	object30	38.383
object23	33.5001	object59	28.729	object7	39.8166	object22	39.173
object46	33.7473	object43	28.7431	object4	41.3496	object5	40.143
object53	33.7723	object15	28.7694	object5	42.0924	object26	45.0875
object47	33.9545	object58	28.9462	object25	42.3309	object3	45.7314
object52	34.1799	object48	29.1438	object28	44.0638	object16	46.6027

Para el caso de las figuras seudo-aleatorias, el lector puede observar que los objetos pueden ser figuras geométricas abiertas (no sólidas), dichos objetos no pueden ser comparados con métodos volumétricos.

Las tablas B y N, son extraídas de [Angeles05d], dichas tablas son ejemplificadas en la figura 54, como se puede observar, abstraer las entidades tridimensionales como secuencias representativas permite la detección de pequeños y grandes cambios entre un par de entidades.

Aunque los resultados obtenidos con la técnica de indexado LB_Keogh muestra ser una *adecuada* aproximación a la búsqueda secuencial. Se realizaron pruebas experimentales con la técnica LB_Kim, ver detalles en el capítulo anterior.

Tabla 11. Distancias DTW entre figuras seudo-aleatorias$_2$.

object28		*object32*		*object38*		*object42*	
object28	0	object32	0	object38	0	object42	0
object29	25.5119	object31	25.5789	object35	35.562	object41	0.009762
object27	35.7427	object33	40.0336	object22	39.7679	object40	6.316
object25	36.3776	object21	41.5871	object37	45.2066	object43	6.4182
object26	39.6253	object22	43.5055	object39	45.2901	object47	7.5454
object20	44.0638	object15	43.5238	object17	46.6745	object46	8.518
object30	49.743	object17	44.2674	object23	47.6049	object45	9.5732
object24	50.1987	object5	44.6061	object21	51.1489	object50	10.806
object6	55.3796	object19	44.964	object0	51.7724	object44	12.2648
object21	57.1548	object18	44.9816	object27	54.4453	object49	13.0753
object19	58.3083	object4	49.7267	object18	54.4786	object52	13.9428
object18	58.379	object23	50.6915	object19	54.5051	object51	15.572
object11	62.2127	object0	52.0521	object36	55.7372	object48	16.1898
object0	62.7515	object35	52.3858	object6	56.6077	object53	16.9134
object7	62.8353	object29	53.0621	object7	56.7608	object59	17.7129
object8	62.9283	object24	54.172	object5	56.7707	object58	18.3248
object22	65.0312	object9	54.2713	object1	58.191	object61	18.5083
object31	65.2103	object20	55.6629	object30	58.2144	object60	19.5365
object35	65.5084	object16	56.2135	object2	58.2171	object57	22.0352
object32	65.7092	object10	56.2931	object24	58.7432	object56	24.4026

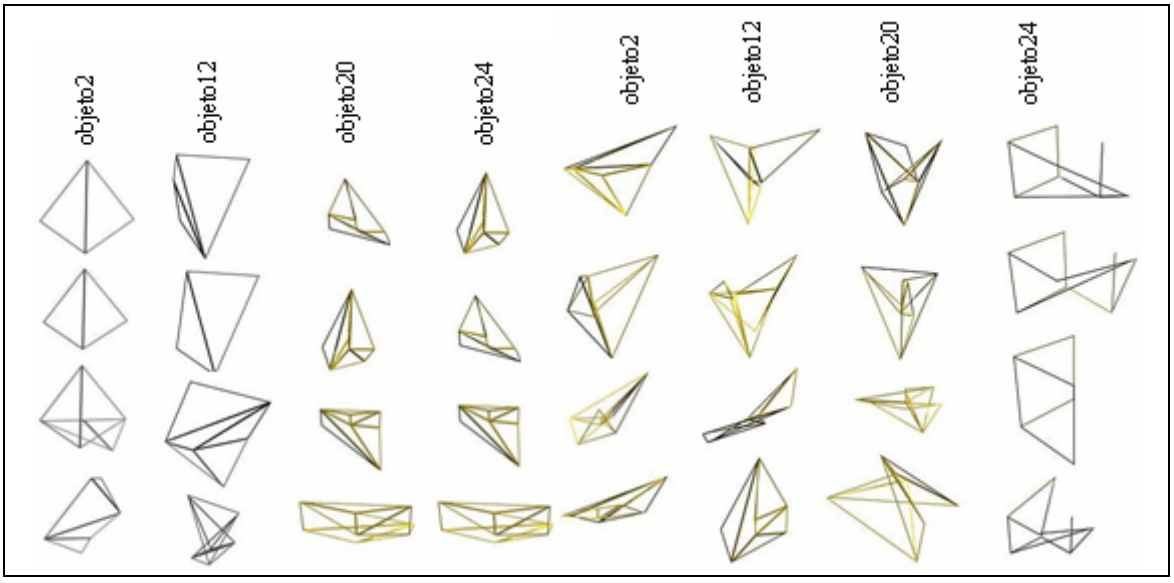


Figura 54. Distancias DTW sobre figuras seudo-aleatorias.

A continuación se muestran resultados de búsqueda secuencial y búsqueda indexada con la técnica LB_Kim, detalles sobre la técnica se encuentran en el capítulo anterior. Los resultados se presentan de la siguiente forma: a)una tabla de distancias para los objetos referentes (Amortiguador01, ChafCil01, Ext_C01, Ext_L01) **colocados en la primer fila de la tabla.** Debajo de los objetos referentes se encuentran las menores distancias con respecto a los demás miembros de la base de datos. Y b)comparar dicha búsqueda secuencial con la búsqueda indexada.

Tabla 12. Distancias DTW y otros objetos.

	Amortiguador02		***ChafCil01***		***Ext_C01***		***Ext_L01***
Amortiguador02	0	ChafCil01	0	Ext_C01	0	Ext_L01	0
Amortiguador01	118.861	ChafCil02	106.2858	Ext_C02	6.672	Ext_L05	5.0388
Muelle01	227.2536	Gengon05	120.635	Ext_C03	7.5414	Ext_L03	5.7651
Cono02	235.3789	Gengon03	123.7277	Ext_C05	9.3554	Ext_L04	11.6447
Ext_L02	237.133	ChafCil03	126.5888	Ext_C04	9.7154	Ext_L02	12.33

Ahora en la siguiente tabla se presentan consultas (búsquedas indexadas). Con k vecinos cercanos, donde k = 5.

Tabla 13. K vecinos cercanos y otros objetos.

Consulta	**Amortiguador01**	**Consulta**	**ChafCil01**	**Consulta**	**Ext_C01**	**Consulta**	**Ext_L01**
Amortiguador01	0	ChafCil01	0	Ext_C01	0	Ext_L01	0
Amortiguador02	0.212	ChafCil02	0.5	Ext_C04	0.03031	Ext_L05	0.23
Cono02	0.5074	ChafCil04	0.6252	Ext_C05	0.12	Ext_L03	0.33
Muelle01	0.98356	ChafCil05	0.6252	Ext_C06	0.251	Ext_L04	0.33
Cono01	1.0586	ChafCil03	0.6252	Ext_C03	0.4365	Ext_L02	0.33

Estos últimos resultados fueron presentados en [Angeles05e].

9.3 DISCUSIÓN

Aunque múltiples técnicas han sido desarrolladas para la comparación de entidades tridimensionales (ver capítulo 2), todas adolecen de algo en particular, las características que distinguen a dos entidades son locales o globales, o en otro sentido más simple, de grano.

Los resultados experimentales de este trabajo muestran que los métodos propuestos tienen gran capacidad para detectar grandes diferencias así como pequeñas diferencias.

Uno de los aspectos más importantes en los conjuntos de objetos sometidos a **cualquier** método de comparación, es el orden del índice de vértice. En [Ming] se propone una heurística para re-orden el índice de vértices. El criterio de selección de esta heurística puede ser variable, ello dependerá del tipo de entidades que se desee comparar.

Aunque no es objetivo principal de este trabajo hacer un estudio sobre las técnicas de indexado, cabe señalar que la selección de dicha técnica debe estar apegada a la comprensión del *comportamiento o naturaleza* de las secuencias.

Es parte de esta discusión argumentar con pruebas experimentales, la posibilidad de nuevas técnicas de similitud, en las que las etapas de construcción, selección y/o extracción de características sean discrecionales, es decir, técnicas de similitud sin ninguna de estas etapas, ya que la dicha etapa asegura que se conoce a priori lo ***relevante*** de un par de entidades.

REFERENCIAS DEL CAPÍTULO

[Angeles05a] A. Angeles-Yreta, J. Figueroa-Nazuno. "Semejanza entre Objetos Tridimensionales aplicando Redes Neuronales Fuzzy ART". *Advances in Computer Science in Mexico*, *Research on Computer Science*. Vol. 13, A. Gelbukh y H. Calvo (Eds.), ISSN: 1665-9899, 2005, pp. 127-134

[Angeles05b] A. Angeles-Yreta, J. Figueroa-Nazuno. "Similarity Search in 3D Objects and Dynamic Time Warping". *Proceedings of International Seminar on Computational Intelligence*. IEEE CIS Mexico Chapter. 2005.

[Angeles05c] A. Angeles-Yreta, J. Figueroa-Nazuno. "Búsqueda de Semejanza entre Objetos Tridimensionales por Indexado". *Congreso Nacional de Física 2005*. ISSN 0187-4713. 2005. pp. 97

[Angeles05d] A. Angeles-Yreta, J. Figueroa-Nazuno. "Computing Similarity Among 3D Objects Using Dynamic Time Warping". *Lecture Notes in Computer Science* (Springer-Verlag), Progress in Pattern Recognition, Image Analysis and Applications: *10th Iberoamerican Congress on Pattern Recognition, CIARP 2005*, Havana, Cuba, November 15-18, 2005, ISSN: 0302-9743, ISSN: 0302-9743. pp. 319-326.

[Angeles05e] A. Angeles-Yreta, J. Figueroa-Nazuno, K. Ramírez-Amaro. "Búsqueda de Semejanza entre Objetos 3D por Indexado". *Decimosexta Reunión de Otoño Comunicaciones, Computación* ROC&C 2005. 2005.

[Ming] Ming-Ching L., Chong-Juch L., Hon-Son D. "A Neural Network Approach to 3-D object identification and pose estimation". *IEEE International Joint Conference on Neural Networks*. pp. 2600-2605.

10 CONCLUSIONES

La *adecuada representación* de problemas afecta en gran medida a la comprensión de los mismos, y de forma consecuente, a la posibilidad de construir soluciones inteligentes por métodos computacionales.

El *problema de semejanza* se presenta en cualquier situación en la que se requiere comparar objetos multidimensionales.

Resulta imposible la *comparación de métodos* para computar semejanza cuando éstos difieren en su aproximación al problema.

Se presentaron dos nuevas formas para representar y determinar semejanza entre poliedros. La primera, no contempla etapas de extracción/selección de características; representa poliedros en *secuencias acopladas* y determina la pertenencia de un poliedro a una clase por medio de una *RNA Fuzzy ART* y distancia de *Hamming*. La segunda, representa poliedros en *secuencias*, y determina distancias de semejanza usando *Dynamic Time Warping*. Ambas de bajo costo computacional. Adicionalmente, se mostró técnicas de indexado para poliedros capaces de aproximarse a una búsqueda secuencial.

La mayoría de los métodos para computar semejanza utilizan al menos una etapa de extracción y/o selección de características, este importante recurso debe ser discrecional. Los resultados de este trabajo indican que, la aplicación de técnicas de extracción y/o selección de características debe estar en función de:

a) La identificación de los *elementos significativos* para un problema dado.
b) Las *propiedades* de las técnicas, métricas, etc., que componen el método para computar semejanza.

Este primer experimento esquematizó la idea de representar y determinar semejanza entre poliedros de tamaño similar.

El problema de semejanza es importante porque se presenta cuando se *comparan entidades* que *no* tienen una *relación de orden* preestablecida, pero que contradictoriamente los criterios de comparación no pueden crear dicho orden.

En el problema de semejanza, no hay un criterio de comparación universal. No existe un método para determinar el error en el problema de semejanza.

Las teorías de la medida sugieren la posibilidad de sustituir la función de semejanza en el paradigma clásico, por un criterio de comparación.

Los descriptores de propiedades, no sólo “representan” una entidad en un espacio menor, sino que adecuan la entidad al criterio de comparación.

En el problema de computar la semejanza entre entidades no siempre se debe incluir alguna etapa de selección/extracción/construcción de características.

No existen pasos generales para el problema de computar semejanza entre entidades.

ANEXO A. CÓDIGO FUENTE

FuzzyART.h: interface for the CFuzzyART class.

```
//////////////////////////////////////////////////////////////////////

//number of input units (F1 layer)
#define M 5

//number of cluster units (F2 layer)
#define N 6


class CFuzzyART
{
  public:
  CFuzzyART(){};
  virtual ~CFuzzyART(){};
  void InitializeFuzzyART(double Rou, double Alpha, double Beta);
  void Input(double*a);       //input the pattern a
  void Step8();
  void Step7();
  void Step6();
  void Step5();
  void Step4();
  void Step3();
  void Step2();               //main loop for training
  void Test();                //testing module for input
  double A[2*M];              //the vector of F0 layer (a,ac)
  double Xa[2*M];             //the vector of F1a layer
  double Ya[N];               //the vector of F2a layer
  double rou;                 //the vigilance parameter [0,1]
  double alpha;               //the choice parameter >0
  double beta;                //learning rate [0,1]
  double m_pWeights[2*M][N]; //from F1 layer(M units) to F2
                              //layer(N units)
  bool ifReset;               // the reset bool parameter
  int J;                      //the unit YJ with the largest signal
};
//#endif
```

FuzzyART.cpp: implementation of the CFuzzyART class.

```
////////////////////////////////////////////////////////////////////////

#include <memory.h>
#include <string.h>
#include "FuzzyART.h"

//initial the parameters of Fuzzy ART
void CFuzzyART::InitializeFuzzyART(double Rou, //[0,1]
                                   double Alpha, //>0
                                   double Beta //[0,1]
                                               )
{
  rou=Rou;
  alpha=Alpha;
  beta=Beta;
  int M2;

  M2 = M*2;

  //initialize the weight matrix
  for(int i=0;i<M2;i++)
    for(int j=0;j<N;j++)
      m_pWeights[i][j]=1; //set every weight to 1

  //initialize the activation of the net's units
  memset(A,0,sizeof(double)*M2);
  memset(Xa,0,sizeof(double)*M2);
  memset(Ya,0,sizeof(double)*N);
  ifReset=false;
  J=0;
}

// Set the rhou value
void CFuzzyART::SetFuzzyART(double Rou) {
      rou=Rou;
}

//input the pattern a and complement coding
void CFuzzyART::Input(double *a) {
  for(int i=0;i<M;i++) {
    A[i]=a[i];
    A[i+M]=1-a[i];
  }
}

// main loop for training of input patterns
void CFuzzyART::Step2() {
  Step3();
  Step4();
  Step5();
  Step8();
}

void CFuzzyART::Step3() {
  int M2;
  M2 = M*2;
  //from F0a to F1a
  //Update the F1
  for(int i=0;i<M2;i++)
    Xa[i]=A[i];
```

```
}

void CFuzzyART::Step4() {
  //calculate signals to F2a units
  double tempw=0;
  double tempIw=0;
  int M2;

  M2 = M*2;
  for(int j=0;j<N;j++) {
    tempw=0;
    tempIw=0;
    for(int i=0;i<M2;i++) {
      tempw+=m_pWeights[i][j];
      if(Xa[i]>m_pWeights[i][j])
        tempIw+=m_pWeights[i][j];
      else
        tempIw+=Xa[i];//where Xa[i]=A[i]=I[i]
    }
    Ya[j]=tempIw/(alpha+tempw);
  }
}

void CFuzzyART::Step5() {
  do {      //loop to find wining the category
    Step6();
    Step7();
  }
  while(ifReset);
}

void CFuzzyART::Step6() {
  J=0;
  double max=0;
  max=Ya[J];
  for(int j=0;j<N;j++) {
    if(max<Ya[j]) {
      J=j;
      max=Ya[J];
    }
  }
}

void CFuzzyART::Step7() {
  double temp=0;
  int M2;

  M2=M*2;
  for(int i=0;i<M2;i++) {
    if(A[i]>m_pWeights[i][J])
      Xa[i]=m_pWeights[i][J];
    else
      Xa[i]=A[i];
    temp+=Xa[i];
  }

  //check reset
  if(temp>=rou*M)
    ifReset=false;
  else {
    ifReset=true;
```

```
    Ya[J]=0;
  }
}

void CFuzzyART::Step8() {
  int M2;
  M2 = M*2;

  for(int i=0;i<M2;i++) {
    m_pWeights[i][J]=
    beta*Xa[i]+(1-beta)*m_pWeights[i][J];
  }
}

// modules for testing of input patterns
void CFuzzyART::Test() {
  Step3();
  Step4();
  Step6();
}
```

FDTW.h: interface for the CFDTW class.

```
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_FDTW_H__E40442EC_AEFC_4360_B973_3B5F20B49678__INCLUDED_)
#define AFX_FDTW_H__E40442EC_AEFC_4360_B973_3B5F20B49678__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <afxtempl.h>
#include <math.h>

class CFDTW
{
public:
      double FDTW();
      bool SetSequences(char *fileQ, char *fileC);
      CFDTW();
      virtual ~CFDTW();

private:
      CArray<double, double> C;
      CArray<double, double> Q;
      double **Matrix;
};

#endif //
!defined(AFX_FDTW_H__E40442EC_AEFC_4360_B973_3B5F20B49678__INCLUDED_)
```

FDTW.cpp: implementation of the CFDTW class.

```
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "FastDynamicTimeWarping.h"
#include "FDTW.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CFDTW::CFDTW()
{

}

CFDTW::~CFDTW()
{

}

bool CFDTW::SetSequences(char *fileQ, char *fileC)
{
      CFile fpQ, fpC;              // Manejadores de archivo (MFC)
      CString token;
      char seps[] = "\n";

// Obtener del archivo fileQ y fileC, la secuencia Q y C //
respectivamente
      if( (fpQ.Open(fileQ, CFile::modeRead, NULL)) && (fpC.Open(fileC,
CFile::modeRead, NULL)) )
      {
            char *bufferQ = new char[fpQ.GetLength()];
            char *bufferC = new char[fpC.GetLength()];

            fpQ.Read(bufferQ, fpQ.GetLength());
            fpC.Read(bufferC, fpC.GetLength());

            token = strtok(bufferQ, seps);

            while(token != "")
            {
                  Q.Add(atof(token));
                  token = strtok(NULL, seps);
            }

            token = strtok(bufferC, seps);

            while(token != "")
            {
                  C.Add(atof(token));
                  token = strtok(NULL, seps);
            }

            fpQ.Close();
```

```
            fpC.Close();

            return true;
      }
      else
            return false;
}

double CFDTW::FDTW()
{
  // Construir la tabla de DTW
  Matrix = new double*[Q.GetSize()];

  for(int i=0; i < Q.GetSize(); i++)
    Matrix[i] = new double [C.GetSize()];

  // Primer elemento de la Matrix
  Matrix[0][0] = sqrt(pow(Q.ElementAt(0)-C.ElementAt(0), 2));

  // Construir la primer fila de la tabla DTW
  for(int j=1; j<C.GetSize(); j++)
    Matrix[0][j] = Matrix[0][j-1] + sqrt(pow(Q.ElementAt(0)-
                                        C.ElementAt(j), 2));

  // Construir la primer columna de la tabla DTW
  for(i=1; i<Q.GetSize(); i++)
    Matrix[i][0] = Matrix[i-1][0] + sqrt(pow(Q.ElementAt(i)-
                                        C.ElementAt(0), 2));

  // Construir el resto de FILAS de la tabla DTW
  for(i=1; i<Q.GetSize(); i++)
     for(j=1; j<C.GetSize(); j++)
     {
       //Obtener la distancia entre puntos (Q y C)
       double pdistance = sqrt(pow(Q.ElementAt(i)-C.ElementAt(j), 2));

       Matrix[i][j] = Matrix[i-1][j-1] + pdistance;

       if ( (Matrix[i-1][j]+pdistance) < Matrix[i][j] )
         Matrix[i][j] = Matrix[i-1][j] + pdistance;

       if ( Matrix[i][j-1]+pdistance < Matrix[i][j])
         Matrix[i][j] = Matrix[i][j-1] + pdistance;
      }
      return Matrix[Q.GetSize()-1][C.GetSize()-1];
}
```

DTW.cpp: implementation of the CDTW class.

```
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "DynamicTimeWarping.h"
#include "DTW.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CDTW::CDTW()
{
	count = 1;
}

CDTW::~CDTW()
{

}

bool CDTW::SetSequences(char *fileQ, char *fileC)
{
	CFile fpQ, fpC;			// Manejadores de archivo (MFC)
	CString token;
	char seps[] = "\n";

	// Obtener del archivo fileQ y fileC, la secuencia Q y C
respectivamente
	if( (fpQ.Open(fileQ, CFile::modeRead, NULL)) && (fpC.Open(fileC,
CFile::modeRead, NULL)) )
	{
		char *bufferQ = new char[fpQ.GetLength()];
		char *bufferC = new char[fpC.GetLength()];

		fpQ.Read(bufferQ, fpQ.GetLength());
		fpC.Read(bufferC, fpC.GetLength());

		token = strtok(bufferQ, seps);

		while(token != "")
		{
			Q.Add(atof(token));
			token = strtok(NULL, seps);
		}

		token = strtok(bufferC, seps);

		while(token != "")
		{
			C.Add(atof(token));
			token = strtok(NULL, seps);
		}

		fpQ.Close();
```

```
            fpC.Close();

            return true;
      }
      else
            return false;
}

double CDTW::DTW(int i, int j)
{
      count++;

      if ( i==Q.GetSize() && j==C.GetSize() )
            return 0;
      else if ( (i==Q.GetSize()) | (j==C.GetSize()) )
            return 1.79769e+308;
      else
      {
            return sqrt(pow((Q.ElementAt(i)-C.ElementAt(j)),2)) +
                  Minimum(DTW(i,j+1), DTW(i+1,j), DTW(i+1,j+1));
      }
}

double CDTW::Minimum(double x, double y, double z)
{
      if ( (x <= y) && (x <= z) )
            return x;
      else if( (y <= x) && (y <= z))
            return y;
      else
            return z;
}
```

DTW.h: interface for the CDTW class.

```
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_DTW_H__CDA06406_6797_483F_AF39_9564EF66463A__INCLUDED_)
#define AFX_DTW_H__CDA06406_6797_483F_AF39_9564EF66463A__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <afxtempl.h>
#include <math.h>

class CDTW
{
public:
	double DTW(int i, int j);
	bool SetSequences(char *fileQ, char *fileC);
	CDTW();
	virtual ~CDTW();
	int count;

private:
	double Minimum(double x, double y, double z);
	CArray<double, double> C;
	CArray<double, double> Q;
};

#endif //
!defined(AFX_DTW_H__CDA06406_6797_483F_AF39_9564EF66463A__INCLUDED_)
```

ANEXO A. CÓDIGO FUENTE

FuzzyART.h: interface for the CFuzzyART class.

```
//////////////////////////////////////////////////////////////////////

//number of input units (F1 layer)
#define M 5

//number of cluster units (F2 layer)
#define N 6


class CFuzzyART
{
  public:
  CFuzzyART(){};
  virtual ~CFuzzyART(){};
  void InitializeFuzzyART(double Rou, double Alpha, double Beta);
  void Input(double*a);       //input the pattern a
  void Step8();
  void Step7();
  void Step6();
  void Step5();
  void Step4();
  void Step3();
  void Step2();               //main loop for training
  void Test();                //testing module for input
  double A[2*M];              //the vector of F0 layer (a,ac)
  double Xa[2*M];             //the vector of F1a layer
  double Ya[N];               //the vector of F2a layer
  double rou;                 //the vigilance parameter [0,1]
  double alpha;               //the choice parameter >0
  double beta;                //learning rate [0,1]
  double m_pWeights[2*M][N]; //from F1 layer(M units) to F2
                              //layer(N units)
  bool ifReset;               // the reset bool parameter
  int J;                      //the unit YJ with the largest signal
};
//#endif
```

FuzzyART.cpp: implementation of the CFuzzyART class.

```
////////////////////////////////////////////////////////////////////////

#include <memory.h>
#include <string.h>
#include "FuzzyART.h"

//initial the parameters of Fuzzy ART
void CFuzzyART::InitializeFuzzyART(double Rou, //[0,1]
                                   double Alpha, //>0
                                   double Beta //[0,1]
                                               )
{
  rou=Rou;
  alpha=Alpha;
  beta=Beta;
  int M2;

  M2 = M*2;

  //initialize the weight matrix
  for(int i=0;i<M2;i++)
    for(int j=0;j<N;j++)
      m_pWeights[i][j]=1; //set every weight to 1

  //initialize the activation of the net's units
  memset(A,0,sizeof(double)*M2);
  memset(Xa,0,sizeof(double)*M2);
  memset(Ya,0,sizeof(double)*N);
  ifReset=false;
  J=0;
}

// Set the rhou value
void CFuzzyART::SetFuzzyART(double Rou) {
      rou=Rou;
}

//input the pattern a and complement coding
void CFuzzyART::Input(double *a) {
  for(int i=0;i<M;i++) {
    A[i]=a[i];
    A[i+M]=1-a[i];
  }
}

// main loop for training of input patterns
void CFuzzyART::Step2() {
  Step3();
  Step4();
  Step5();
  Step8();
}

void CFuzzyART::Step3() {
  int M2;
  M2 = M*2;
  //from F0a to F1a
  //Update the F1
  for(int i=0;i<M2;i++)
    Xa[i]=A[i];
```

```
}

void CFuzzyART::Step4() {
  //calculate signals to F2a units
  double tempw=0;
  double tempIw=0;
  int M2;

  M2 = M*2;
  for(int j=0;j<N;j++) {
    tempw=0;
    tempIw=0;
    for(int i=0;i<M2;i++) {
      tempw+=m_pWeights[i][j];
      if(Xa[i]>m_pWeights[i][j])
        tempIw+=m_pWeights[i][j];
      else
        tempIw+=Xa[i];//where Xa[i]=A[i]=I[i]
    }
    Ya[j]=tempIw/(alpha+tempw);
  }
}

void CFuzzyART::Step5() {
  do {      //loop to find wining the category
    Step6();
    Step7();
  }
  while(ifReset);
}

void CFuzzyART::Step6() {
  J=0;
  double max=0;
  max=Ya[J];
  for(int j=0;j<N;j++) {
    if(max<Ya[j]) {
      J=j;
      max=Ya[J];
    }
  }
}

void CFuzzyART::Step7() {
  double temp=0;
  int M2;

  M2=M*2;
  for(int i=0;i<M2;i++) {
    if(A[i]>m_pWeights[i][J])
      Xa[i]=m_pWeights[i][J];
    else
      Xa[i]=A[i];
    temp+=Xa[i];
  }

  //check reset
  if(temp>=rou*M)
    ifReset=false;
  else {
    ifReset=true;
```

```
    Ya[J]=0;
  }
}

void CFuzzyART::Step8() {
  int M2;
  M2 = M*2;

  for(int i=0;i<M2;i++) {
    m_pWeights[i][J]=
    beta*Xa[i]+(1-beta)*m_pWeights[i][J];
  }
}

// modules for testing of input patterns
void CFuzzyART::Test() {
  Step3();
  Step4();
  Step6();
}
```

FDTW.h: interface for the CFDTW class.

```
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_FDTW_H__E40442EC_AEFC_4360_B973_3B5F20B49678__INCLUDED_)
#define AFX_FDTW_H__E40442EC_AEFC_4360_B973_3B5F20B49678__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <afxtempl.h>
#include <math.h>

class CFDTW
{
public:
      double FDTW();
      bool SetSequences(char *fileQ, char *fileC);
      CFDTW();
      virtual ~CFDTW();

private:
      CArray<double, double> C;
      CArray<double, double> Q;
      double **Matrix;
};

#endif //
!defined(AFX_FDTW_H__E40442EC_AEFC_4360_B973_3B5F20B49678__INCLUDED_)
```

FDTW.cpp: implementation of the CFDTW class.

```
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "FastDynamicTimeWarping.h"
#include "FDTW.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CFDTW::CFDTW()
{

}

CFDTW::~CFDTW()
{

}

bool CFDTW::SetSequences(char *fileQ, char *fileC)
{
      CFile fpQ, fpC;              // Manejadores de archivo (MFC)
      CString token;
      char seps[] = "\n";

// Obtener del archivo fileQ y fileC, la secuencia Q y C //
respectivamente
      if( (fpQ.Open(fileQ, CFile::modeRead, NULL)) && (fpC.Open(fileC,
CFile::modeRead, NULL)) )
      {
            char *bufferQ = new char[fpQ.GetLength()];
            char *bufferC = new char[fpC.GetLength()];

            fpQ.Read(bufferQ, fpQ.GetLength());
            fpC.Read(bufferC, fpC.GetLength());

            token = strtok(bufferQ, seps);

            while(token != "")
            {
                  Q.Add(atof(token));
                  token = strtok(NULL, seps);
            }

            token = strtok(bufferC, seps);

            while(token != "")
            {
                  C.Add(atof(token));
                  token = strtok(NULL, seps);
            }

            fpQ.Close();
```

```
            fpC.Close();

            return true;
      }
      else
            return false;
}

double CFDTW::FDTW()
{
  // Construir la tabla de DTW
  Matrix = new double*[Q.GetSize()];

  for(int i=0; i < Q.GetSize(); i++)
    Matrix[i] = new double [C.GetSize()];

  // Primer elemento de la Matrix
  Matrix[0][0] = sqrt(pow(Q.ElementAt(0)-C.ElementAt(0), 2));

  // Construir la primer fila de la tabla DTW
  for(int j=1; j<C.GetSize(); j++)
    Matrix[0][j] = Matrix[0][j-1] + sqrt(pow(Q.ElementAt(0)-
                                        C.ElementAt(j), 2));

  // Construir la primer columna de la tabla DTW
  for(i=1; i<Q.GetSize(); i++)
    Matrix[i][0] = Matrix[i-1][0] + sqrt(pow(Q.ElementAt(i)-
                                        C.ElementAt(0), 2));

  // Construir el resto de FILAS de la tabla DTW
  for(i=1; i<Q.GetSize(); i++)
     for(j=1; j<C.GetSize(); j++)
     {
       //Obtener la distancia entre puntos (Q y C)
       double pdistance = sqrt(pow(Q.ElementAt(i)-C.ElementAt(j), 2));

       Matrix[i][j] = Matrix[i-1][j-1] + pdistance;

       if ( (Matrix[i-1][j]+pdistance) < Matrix[i][j] )
         Matrix[i][j] = Matrix[i-1][j] + pdistance;

       if ( Matrix[i][j-1]+pdistance < Matrix[i][j])
         Matrix[i][j] = Matrix[i][j-1] + pdistance;
      }
      return Matrix[Q.GetSize()-1][C.GetSize()-1];
}
```

DTW.cpp: implementation of the CDTW class.

```
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "DynamicTimeWarping.h"
#include "DTW.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CDTW::CDTW()
{
	count = 1;
}

CDTW::~CDTW()
{

}

bool CDTW::SetSequences(char *fileQ, char *fileC)
{
	CFile fpQ, fpC;			// Manejadores de archivo (MFC)
	CString token;
	char seps[] = "\n";

	// Obtener del archivo fileQ y fileC, la secuencia Q y C
respectivamente
	if( (fpQ.Open(fileQ, CFile::modeRead, NULL)) && (fpC.Open(fileC,
CFile::modeRead, NULL)) )
	{
		char *bufferQ = new char[fpQ.GetLength()];
		char *bufferC = new char[fpC.GetLength()];

		fpQ.Read(bufferQ, fpQ.GetLength());
		fpC.Read(bufferC, fpC.GetLength());

		token = strtok(bufferQ, seps);

		while(token != "")
		{
			Q.Add(atof(token));
			token = strtok(NULL, seps);
		}

		token = strtok(bufferC, seps);

		while(token != "")
		{
			C.Add(atof(token));
			token = strtok(NULL, seps);
		}

		fpQ.Close();
```

```
            fpC.Close();

            return true;
      }
      else
            return false;
}

double CDTW::DTW(int i, int j)
{
      count++;

      if ( i==Q.GetSize() && j==C.GetSize() )
            return 0;
      else if ( (i==Q.GetSize()) | (j==C.GetSize()) )
            return 1.79769e+308;
      else
      {
            return sqrt(pow((Q.ElementAt(i)-C.ElementAt(j)),2)) +
                  Minimum(DTW(i,j+1), DTW(i+1,j), DTW(i+1,j+1));
      }
}

double CDTW::Minimum(double x, double y, double z)
{
      if ( (x <= y) && (x <= z) )
            return x;
      else if( (y <= x) && (y <= z))
            return y;
      else
            return z;
}
```

DTW.h: interface for the CDTW class.

```
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_DTW_H__CDA06406_6797_483F_AF39_9564EF66463A__INCLUDED_)
#define AFX_DTW_H__CDA06406_6797_483F_AF39_9564EF66463A__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <afxtempl.h>
#include <math.h>

class CDTW
{
public:
	double DTW(int i, int j);
	bool SetSequences(char *fileQ, char *fileC);
	CDTW();
	virtual ~CDTW();
	int count;

private:
	double Minimum(double x, double y, double z);
	CArray<double, double> C;
	CArray<double, double> Q;
};

#endif //
!defined(AFX_DTW_H__CDA06406_6797_483F_AF39_9564EF66463A__INCLUDED_)
```

REFERENCIAS

[Tversky77] Tversky A. “Features of Similarity”. *In Readings in Cognitive Science*. Morgan Kaufmann 1988. (From Psychological Review 84, 327-352, 1977).

[Lin98] Lin D. “An information-theoretic definition of similarity”. *In Proceedings of the 15th International Conference on Machine Learning*, pp. 196-304, 1998.

[Krantz71] Krantz, D. H., Luce, R. D., Suppes, P., & Tversky, A. (1971). *Foundations of measurement: Vol. 1. Additive and polynomial representations*. New York: Academic.

[Luce90] Luce, R. D., Krantz, D. H., Suppes, P., & Tversky, A. (1990). *Foundations of measurement: Vol. 3. Representation, axiomatization, and invariance*. San Diego, CA: Academic.

[Suppes89] Suppes, P., Krantz, D. H., Luce, R. D., & Tversky, A. (1989). *Foundations of measurement: Vol. 2. Geometrical, threshold, and probabilistic representations*. San Diego, CA: Academic.

[Guzmán68] Guzmán A. “Computer Recognition of Three-dimensional Objects in a Visual Scene”. Ph. D. Thesis, Electrical Engineering Department, M. I. T., December 1968. Also available as Project MAC Technical Report MAC TR 59. AD-692-200.

[Roberts65] Roberts L.G. “Machine perception of three-dimensional solids”. *Optical an Electrooptical information processing*, pp. 159-197, J T Tippett et al (eds) MIT Press 1965.

[Shilane04] Shilane P., Kazhdan M., Min P., Funkhouser T. “The Princeton Shape Benchmark”. *In Proceedings of Shape Modeling International*, 2004.

[Brennecke04] Brennecke A., Isenberg T. “3D Shape Matching Using Skeleton Graphs”. *In Proceedings of Simulation und Visualisierung*, pp. 299-310, 2004

[Cyr01] Cyr C. M., Kimia B. “3D Object Recognition Using Shape Similarity-Based Aspect Graph”. *In Eighth International Conference on Computer Vision* (ICCV-01), pp. 254-261, 2001.

[Osada01] Osada R., Funkhouser T., Chazelle B., Dobkin D. “Matching 3D models with shape distributions”. pp. 154-166, 2001.

[Hlavatý03] Hlavatý T., Skala V. “A Survey of Methods for 3D Model Feature Extraction”. *Bulletin of IV. Seminar Geometry and Graphics in Teaching Contemporary Engineer*, No: 13/03, pp. 5-8, 2003.

[Boyer02] Boyer K. L., Srikantiah R., Flynn P. J. “Salience Sequential Surface Organization for Free-Form Object Recognition”. *Computer Vision and Image Understanding*, Vol. 88, No. 3, 2002.

[Mokhtarian01] Mokhtarian F., Khalili N., Yuen P. “Multi-scale free-form 3D object recognition using 3D models”. *Image and Vision Computing*, Vol. 19, pp. 271-281, 2001.

[Quek01]Quek K. H., Yarger R. W. I., Kirbas C. “Surface Parameterization in Volumetric Images for Curvature-based Feature Classification”. *IEEE Transactions on Systems*, Man and Cybernetics. 2001.

[Yarger00]Yarger R. W. I., Quek K. H. “Surface Parameterization in Volumetric Images for Feature Classification”. *IEEE Inter. Symposium of Bio-Eng.* BIBE 2000, pp. 297-303, 2000.

[Funkhouser03] Funkhouser T., Min P., Kazhdan M., Chen J., Halderman A., Dobkin D. “A Search Engine for 3D Models”. *ACM Transactions on Graphics*, Vol. 22, Issue 1, pp. 83-105, 2003.

[Tangelder03] Tangelder J. W. H., Veltkamp R. C. “Polyhedral Model Retrieval Using Weighted Point Sets”. *International Journal of Image and Graphics*, Vol. 3, No. 1, pp. 209-229, 2003.

[Hilaga01] Hilaga M., Shinagawa Y., Kohmura T., Kunii T. L. “Topology Matching for Fully Automatic Similarity Estimation of 3D shapes”. *SIGGRAPH 2001*, pp. 203-212, 2001.

[Novotni01] Novotni M., Klein R. “A Geometric Approach to 3D Object Comparison”. *Int. Conf. on Shape Modeling and Applications*, pp. 154-166, 2001.

[Bradski95] Bradski G., Grossberg S. “Fast Learning VIEWNET Architectures for Recognizing 3-D Objects from Multiple 2-D Views”. *Neural Networks Special Issue on ATR*. pp. 1053-1080, 1995.

[Seibert92] Seibert M., Waxman A. M. “Adaptive 3-D Object Recognition from Multple Views”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 14, No. 2, pp. 107-124. 1992.

[Nazuno01] Figueroa-Nazuno, Kashiwamoto-Yabuta J., Vargas-Medina E. “Clasificación y representación de resultados experimentales en 3-D, mediante una red neuronal”. *XXXIV Congreso Nacional de Física*, 1991.

[Ming] Ming-Ching L., Chong-Juch L., Hon-Son D. “A Neural Network Approach to 3-D object identification and pose estimation”. *IEEE International Joint Conference on Neural Networks*. pp. 2600-2605.

[Chung] Chung L. W., Yuan L. F., Kuo T. C., Lingutla T. “A Hierarchical Multiple View Approach to Three Dimensional Object Recognition”. *IEEE Transactions on Neural Networks*, vol. 2, no. 5, pp. 568-576.

[Sossa98] Sossa H., Rayon P. Figueroa J. “2D Object Recognition by Indexing through a modified Art-2 Neural Network”. *The Fourth World Congress on Expert Systems*, 1998.

[Angeles04a] Angeles-Yreta A., Solís-Estrella H., Landassuri-Moreno V., Figueroa-Nazuno J. “Similarity Search In Seismological Signals”. *Fifth Mexican Internacional Conference on Computer Science*. pp. 50-56. 2004.

[Angeles04b] Angeles-Yreta A, Figueroa-Nazuno J. “Análisis de Similitud en Señales Sísmicas con Red Neuronal Artificial Fuzzy ART”. *Congreso Nacional de Física 2004*. ISSN 0187-4713, pp. 114, 2004.

[Angeles05a] A. Angeles-Yreta, J. Figueroa-Nazuno. “Semejanza entre Objetos Tridimensionales aplicando Redes Neuronales Fuzzy ART”. *Advances in Computer Science in Mexico, Research on Computer Science*. Vol. 13, A. Gelbukh y H. Calvo (Eds.), ISSN: 1665-9899, 2005, pp. 127-134.

[Angeles05b] A. Angeles-Yreta, J. Figueroa-Nazuno. “Similarity Search in 3D Objects and Dynamic Time Warping”. *Proceedings of International Seminar on Computational Intelligence*. IEEE CIS Mexico Chapter. 2005.

[Angeles05c] A. Angeles-Yreta, J. Figueroa-Nazuno. “Búsqueda de Semejanza entre Objetos Tridimensionales por Indexado”. *Congreso Nacional de Física 2005*. ISSN 0187-4713. 2005. pp. 97

[Angeles05d] A. Angeles-Yreta, J. Figueroa-Nazuno. “Computing Similarity Among 3D Objects Using Dynamic Time Warping”. *Lecture Notes in Computer Science* (Springer-Verlag), Progress in Pattern Recognition, Image Analysis and Applications: *10th Iberoamerican Congress on Pattern Recognition, CIARP 2005*, Havana, Cuba, November 15-18, 2005, ISSN: 0302-9743, ISSN: 0302-9743. pp. 319-326.

[Angeles05e] A. Angeles-Yreta, J. Figueroa-Nazuno, K. Ramírez-Amaro. “Búsqueda de Semejanza entre Objetos 3D por Indexado”. *Decimosexta Reunión de Otoño Comunicaciones, Computación* ROC&C 2005. 2005.

[Angeles05c] 4. Angeles-Yreta A. et al. “Clasificación y Similitud de Espectros de Respuesta de Aceleración”. *XV Congreso Nacional de Ingeniería Sísmica*. Artículo I-20. 2005.

[Heins95] Lucien G. Heins and Daniel R. Tauritz. Adaptive resonance theory (art): An introduction. Technical Report 95-35, Leiden University, 1995.

[Grossberg76a] Grossberg S. “Adaptive pattern classification and universal recoding. I. Parallel development and coding of neural feature detectors”. Biol. Cybernet 23, pp. 121-134, 1976.

[Grossberg76b] Grossberg S. “Adaptive pattern classification and universal recoding. II. Feedback, expectation, olfacation, and illusions”. Biol. Cybernet. 23, pp.187-202, 1976.

[Carpenter87a] Carpenter G., Grossberg S. “A Massively Parallel Architecture for a Self Organizing Neural Pattern Recognition Machine”. *Computer Vision, Graphics, and Image Processing*, Vol. 37, pp.54-115, 1987.

[Carpenter87b] Carpenter G., Grossberg S. “ART 2: Self-organization of stable category recognition codes for analog input patterns”. *Applied Optics* 26(23): pp. 4919-4930, 1987.

[Carpenter91a] Carpenter G., Grossberg S., Rosen D.B. “ART2-A: An Adaptive Resonance Algorithm for Rapid Category Learning and Recognition”. *Neural Networks*, Vol. 4, pp.493-504, 1991.

[Carpenter91b] Carpenter G., Grossberg S., Reynolds J. H. "ARTMAP: Supervised Real-Time Learning and Classification of Nonstationary Data by a Self-Organizing Neural Network". *Neural Networks*, Vol. 4, pp.565-588, 1991.

[Carpenter91c] Carpenter G., Grossberg S. "Fuzzy ART: Fast Stable Learning and Categorization of Analog Patterns by an Adaptive Resonance System". *Neural Networks*, Vol. 4, pp.759-771, 1991.

[Carpenter90] Carpenter G., Grossberg S. "ART3: Hierarchical Search Using Chemical
Transmitters in Self-Organizing Pattern Recognition Architectures". *Neural Networks*, Vol. 3, pp.129-152, 1990.

[Carpenter92] Carpenter G., Grossberg S., Markuzon N., Reynolds J.H., Rosen D.B. "Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps". *IEEE Transactions on Neural Networks*, Vol. 3.No.5, pp.698-713, 1992.

[Koegh02] Keogh E. "Exact Indexing of Dynamic Time Warping". *In Proceedings of the 28th VLDB Conference*, pp. 406-417, 2002.

[Yi98] Yi B. K., Jagadish H. V., Faloutsos C. "Efficient Retrieval of Similar Time Sequences Under Time Warping". *In Proceedings of the 14th International Conference on Data Engineering (ICDE'98)*, pp. 201-208, IEEE Computer Society Press, 1998.

[Deller00] Deller J. R., Hansen J. H. L, Proakis J. G. *Discrete-Time Processing of Speech Signals*. Wiley-IEE Press, ISBN: 0780353862, Second Edition, pp 623-676, 2000.

[Kim01] Kim S. W., Park S., Chu W. W. "An Index-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Database". *In Proceedings of the 17th International Conference on Data Engineering (ICDE'01)*, pp. 607-614, 2001.

[Pandolfi96] Pandolfi F., Oliver M., Wolski M. *Microsoft Foundation Class 4 Bible*. Waite Group Press, ISBN: 1571690212, 1996.

[Berndt96] Berndt D. J., Clifford J. "Finding Patterns in Time Series: A Dynamic Programming Approach". *Advances in Knowledge Discovery and Data Mining*, AAAI/MIT, pp. 229-248, 1996.

[Guttman84] Guttman A. “R-Trees: A Dynamic Index Structure for Spatial Searching”. *In Proceedings of ACM SIGMOD’84 Conference on Management of Data*, ACM Press, pp. 47-57, 1984.

[Agrawal93] Agrawal R., Faloutsos C., Swami A. “Efficiency Similarity Search in Sequence Databases”. *In Proceedings of the 4th Conference of Foundations of Data Organization and Algorithms (FODO’93)*, Lecture Notes in Computer Science, pp. 69-84, 1993.

[Chan02] Chan K.P., Fu A., Yu C. “Haar wavelets for efficient similarity search of time-series with and without time warping”. *IEEE Transactions on Knowledge and Data Engineering, to appear*, 2002.

[Korn97] Korn F., Jagadish H., Faloutsos C. “Efficiently supporting ad hoc queries in large datasets of time sequences”. *In Proceedings of ACM SIGMOD’97*, pp. 289-300, 1997.

[Keogh01] Keogh E., Chakrabarti K., Pazzani M., Mehrotra. “Locally adaptive dimensionality reduction for indexing large time series databases”. *In Proceedings of ACM SIGMOD’01 Conference on Management of Data*, pp. 151-162, 2001.

[Morinaka01] Morinaka Y., Yoshikawa M., Amagasa T., Uemura S. “The L-index: An Indexing Structure for Efficient Subsequence Matching in Time Sequence Databases”. *In Proceedings of Pacific-Asian Conference on Knowledge Discovery and Data Mining*, PAKDD’01, pp. 51-60, 2001.

[Lin03] Lin J., Keogh E., Lonardi S., Chiu B. “A Symbolic Representation of Time Series, with Implications for Streaming Algorithms”. *In Proceedings of the 8th ACM SIGMOD’03*, Workshop on Research Issues in Data Mining and Knowledge Discovery. pp. 2-11, 2003.

[Jang00] Jang J.S.R, Gao M.Y. “A Query-by-Singing System based on Dynamic Programming”. *International Workshop on Systems Resolutions* (the 8th Bellman Continuum), pp. 85-89, 2000.

[Hartman96] Hartman, J., Wernecke, J. The VRML 2.0 handbook: building moving worlds on the web, Addison-Wesley, 1996.

[Makoto98] Makoto M., Takuji N. “Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator”. *ACM Transactions on Modeling and Computer Simulation*, ACM Press, Vol. 8, 1998, pp. 3-30.

GLOSARIO

ANN, **A**rtificial **N**eural **N**etwork

ART, **A**daptive **R**esonance **T**heory

CAD, **C**omputer **A**ssisted **D**esign

DAC, **D**iseño **A**sistido por **C**omputadora

DTW, **D**ynamic **T**ime **W**arping

FDTW, Fast Dynamic Time Warping

MFC, **M**icrosoft **F**oundation **C**lass

RNA, **R**ed **N**euronal **A**rtificial